8-11-2020

# Advances in Deep Learning through Gradient Amplification and Applications

Sunitha Basodi

# ADVANCES IN DEEP LEARNING THROUGH GRADIENT AMPLIFICATION AND APPLICATIONS

by

SUNITHA BASODI

Under the Direction of Yi Pan, Ph.D.

## ABSTRACT

Deep neural networks currently play a prominent role in solving problems across a wide variety of disciplines. Improving performance of deep learning models and reducing their training times are some of the ongoing challenges. Increasing the depth of the networks improves performance but suffers from the problem of vanishing gradients and increased training times. In this research, we design methods to address these challenges in deep neural networks and demonstrate deep learning applications in several domains. We propose

a gradient amplification based approach to train deep neural networks, which improves their training and testing accuraries, addresses vanishing gradients, as well as reduces the training time by reaching higher accuracies even at higher learning rates. We also develop an integrated training strategy to enable/disable amplification at certain epochs. Detailed analysis is performed on different neural networks using random amplification, where the layers to be amplified are selected randomly. The implications of gradient amplification on the number of layers, types of layers, amplification factors, training strategies and learning rates are studied in detail. With this knowledge, effective ways to update gradients are designed to perform amplification at layer-level and also at neuron-level. Lastly, we provide applications of deep learning methods to some of the challenging problems in the areas of smartgrids and bioinformatics. Deep neural networks with feed forward architectures are used to solve data integrity attacks in smart grids. We propose an image based preprocessing method to convert heterogenous genomic sequences into images which are then classified to detect Hepatitis C virus(HCV) infection stages. In summary, this research advances deep learning techniques and their applications to real world problems.

INDEX WORDS: Deep learning, Gradient amplification, Learning rate, Vanishing gradient, Smart grid, Sequence image normalization

# ADVANCES IN DEEP LEARNING THROUGH GRADIENT AMPLIFICATION AND APPLICATIONS

by

SUNITHA BASODI

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

in the College of Arts and Sciences

Georgia State University

2020

# ADVANCES IN DEEP LEARNING THROUGH GRADIENT AMPLIFICATION AND APPLICATIONS

by

SUNITHA BASODI

Committee Chair: Yi Pan

Committee: Rafal Angryk
Zhipeng Cai
Pavel Skums
Andrey Shilnikov

Electronic Version Approved:

Office of Graduate Studies
College of Arts and Sciences
Georgia State University
August 2020

## DEDICATION

I would like to devote this to my guru Shri Mataji Nirmala Devi for always guiding and transforming my life into a peaceful journey. I also dedicate this to my parents Basodi Mani kumari and Basodi Satyanarayana for their love, encouragement and immense belief; my loving and ever supporting husband Krishna Pusuluri and our dear son Virata Pusuluri.

# ACKNOWLEDGEMENTS

This research would not have been possible without the support of many people. I would like to thank everyone who has been very supportive and encouraging in my life. I sincerely share my gratitude to Dr. Yi Pan for being my advisor and mentor whose expertise and guidance have constantly encouraged me to perform beyond. He brings so much positive energy and support even at difficult times and has a true desire for students to succeed in life. I thank Dr. Andrey Shilnikov for letting me volunteer in his lab before my Ph.D. and inspiring me to pursue this path. I take this opportunity to thank Dr. Pavel Skums and Dr. Alexandar Zelikovsky who have been amazing collaborators. I revere Dr. Rafal Angryk for his encouragement, timeliness and hardworking nature. I would like to thank all the committee members Dr. Zhipeng Cai, Dr. Pavel Skums, Dr. Rafal Angryk and Dr. Andrey Shilnikov for your constructive comments and suggestions. I thank all the professors at Georgia State University(GSU) who taught computer science and other topics and guided me in strengthening my concepts. A special note of thanks to all the staff at GSU who have supported me in various aspects. I specially commend Tammie Dudley and Paul for always being so patient and available for the students.

I thank all the members of Dr. Pan's research group for their innovative ideas, inspiring discussions and positive criticism during our lab meetings. I feel blessed to have amazing siblings Sujatha Basodi, Anjali Basodi, Nagaraj Basodi and Shivaraj Basodi for their immense love and belief. I thank P.V. Uma Maheswara Rao and Kameswari Sista for their love and support. I thank all the sahaja yogis, especially from Atlanta, who have welcomed and cared for us as a family and guided me in Sahaja Yoga meditation. I thank all my cousins, family and friends in India, Atlanta and else where who have made this journey memorable. I thank Mrs. Katyayani Kunapuli and family for always being there to guide educationally and personally. I feel obliged for the training given by all my teachers from schooling to engineering.

My research was supported by the department of computer science and Molecular Basis of Disease(MBD) fellowship at GSU. Some of the work is funded by the following grants of my advisor and collaborators (Dr. Wenzhan Song, Dr. Pavel Skums and Dr. Alex Zelikovsky): NIH Grant 1R01EB025022 and National Science Foundation(NSF) grants DUE-1303359, DBI-1564899 and CCF-1619110. This work used the Advanced Research Computing Technology and Innovation Core (ARCTIC) resources, which is supported by the National Science Foundation Major Research Instrumentation (MRI) grant number CNS-1920024. We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Tesla K40 GPU also used for this research.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# 1

# INTRODUCTION

Deep learning models have achieved state-of-the-art performances in several areas including computer vision [1–5], automatic speech recognition [6–11], natural language processing [12–16] and beyond [17–24]. In these interdisciplinary applications, deep learning models have produced results comparable or sometimes superior to human experts. Although these acheivements show robustness of deep neural networks, there are several areas where these models could be improved further. Designing new architectures, automatic tuning of network hyperparameters, improving training time of the models, designing efficient functions (activation, kernal and pooling are some of such challenges [25–30].

## 1.1 Motivation

Deep learning models are designed, trained, and tuned to achieve better performance for a given dataset. Their performance improve further with the increase in the depth of the network [31]. Some of the major challenges associated with the increase in the network architecture is the high amount of time required to train the model even on parallel computation resources and vanishing gradients [31]. Training deep neural networks is time-consuming, which could take days or sometimes weeks depending on the type of the model architecture, size of the dataset and hardware resources. One way to speed up the training process is to increase the learning rate. This accelerates the training process by quickly converging to optima, but also has the risk of missing the global optima resulting in sub-optimal solutions or sometimes non-convergence [32]. Lower learning rates does not have such a risk and can converge to optima, but increases training speeds. In general, training process with a learning rate scheduler begins with higher learning rates for a few epochs, followed by reduction of learning rates for the next couple of epochs; which is repeated until the desired optima or

model performance is achieved. Some optimization algorithms automatically determine the learning rates with the epochs dynamically without the requirement of manual intervention. However, these methods also have some fallbacks and do not always converge to optimal solutions. One way to improve the training speed of deep learning models can be to determine ways to achieve optimal model parameters at larger learning rates. There have been multiple efforts to analyze gradients and accordingly modify gradients or learning rates dynamically for weight updates during training process, but there is no detailed analysis on the impact of the modification factor on the performance.

The other important area of research in deep learning models is to prevent vanishing gradient problem [33–35]. The vanishing gradient problem occurs during training of artificial neural networks, specifically during backpropagation. There are several approaches to avoid this problem. One suggested early method was to perform a two step training process which involves network weight initialization followed by fine-tuning using backpropagation method [36]. The other simpler methods that prevent this problem are Rectified Linear Unit (ReLU) activation function [37,38] and batch normalization (BN) [39]. Since ReLU activation saturates inputs in only one direction, therefore has less impact of vanishing gradients. The other recent approach of batch normalization not only improves the performance of the model, but also reduces vanishing gradient problems. Resnet Architecture have residual connections which also overcome vanishing gradient problem to some extent [1]. Lately, due to the improvement of hardware along with the computational abilities of Graphical Processing Units (GPUs), neural networks can be trained without the issue of vanishing gradients to some extent.

Though there have been efforts to address the above mentioned problems independently, there have been minimal efforts to identify an integrated solution to improve the performance of the model by addressing both vanishing gradients and accelerating the training process by achieving higher performance at larger learning rates.

## 1.2 Existing Work

In this section, we briefly discuss the existing approaches to address vanishing gradient problem, reduce the training time of deep learning models, and study the impact of learning rates.

### 1.2.1 Vanishing gradients

Vanishing gradient problem [33–35] occurs while training artificial neural networks during backpropagation and can become significant with the increase of depth of the network. In gradient-based learning methods, during backpropagation, network weights are updated proportional to the gradient value (partial derivative of the cost function with respect to the current weights) after each training iteration (epoch). Depending on the type of the activation functions and network architectures, sometimes the gradient value is too small and its value gets gradually diminished during backpropagation to the initial layers. This prevents the network from updating its weights and also sometimes when the value is too small, the network may be completely stopped from training (updating weights). Though there is no fundamental solution to this problem, but some of the approaches help to avoid it [40]. One such approach consists of performing a two step training process. In the first step, network weights are trained using unsupervised learning methods (such as auto-encoding) and then the weights are fine-tuned using backpropagation method [36]. Other simpler methods that prevent this problem are ReLU activation function [37,38], batch normalization(BN) [39] and Resnet networks [1]. ReLU activation zeros the negative values and only considers positive values. As it saturates inputs in only one direction, it has less impact of vanishing gradients. The other approach, batch normalization, also reduces vanishing gradient problems other than boosting the performance of the model. In batch normalization, during every training iteration, the input data is normalized to reduce its variance, so that the data does not have large bounds. Since inputs are normalized, gradients are also regulated [39]. Resnet network architectures have residual networks have residual connections which help to improve on

this problem. In addition to these approaches, recent advancement in the hardware has also played a crucial role in solving this issue. Increased computational abilities and availability of GPUs aid in reducing this problem. There have been efforts to also overcome this problem by identifying it during the training process. One such method is to have layer specific learning rates computed based on the gradients of the layer during its training [41].

### 1.2.2 Learning rates

Learning rate is one of the most important hyperparameters which controls the performance of deep neural networks. Having higher learning rates cause the model to train faster but might have sub-optimal solutions. However, lower learning rates take longer time to train the model, but can achieve better optimal solutions [32]. There are several approaches designed to take advantage of them. One such method is learning rate scheduler where we start with higher learning rates and gradually lower the rates with training epochs [42]. There are several ways in which such a scheduler can be designed, namely, directly assigning the learning rates to the epochs, gradually decaying the learning rate based on the current learning rate, current epoch and total number of epochs(time-based decay); reducing the learning rate in a step-wise manner after a certain number of epochs(step decay); and exponentially decaying the learning rate based on the initial learning rate and the current epoch(exponential decay). Article [43] summarizes all the above discussed methods in detail. Another paper [44] shows that models can achieve similar test performance without decaying the learning rate but instead by increasing the batch size. This method not only has fewer parameter updates but also increases parallelism thereby reducing training times. Salimans et al. [45] normalize weights while training to speed up the performance of models trained with stochastic gradient descent.

**Adaptive learning rates for layers/neurons/parameters** Another approach to overcome identifying learning rate hyperparameters is by adapting learning rate dynamically based on the performance of the optimization algorithm without need of any scheduling, some

of such methods include Adagrad [46], Adadelta [47], RMSprop [48] and Adam [49]. There have also been several efforts to improve models with adaptive learning rates [50–54]. Ede et al. [55] control the gradients from being propagated when the expected loss is over a defined boundaries. This causes the learning rates to be dynamically addjuste during training. Paper [56] designs a controller to automatically manage learning rate by identifying informative features. Authors of [57] identify learning rates using reinforcement learning based approach by analyzing the training history. Experiments are performed on CIFAR-10 AND FMNIST database have improved performance emphasizing its advantages. You et al. [58] adaptively scale the learning rates of layers to improve the performance of models trained in large batches in parallel and perform experiments on AlexNet [59]. Paper [60] proposes a layer-wise adaptive learning rate computation by using layer weight dependent matching factor which is computed dynamically during training based on the layer type. Authors demonstrate the advantage of their method with mathematical equations and experimental results.

#### 1.2.3 Analysis of gradients

Gradients provide vital information on various aspects such as training progression, weight fluctuations, model convergence and so on. This information can be used to address vanishing/exploding gradients, accelerate the training process or dynamically modify learning rates while training the model. Some of the adaptive learning rate algorithms [46] [47] mentioned above use the gradients of a few iterations to identify a suitable learning rate on the current iteration. Zhang et al. [60] scale the gradients of a layer based on a matching factor which is computed during training based on the weights and type of the layer.

### 1.3 Contributions

This proposal focuses on two main aspects of deep neural networks. Firstly, designing algorithms to address some of the existing challenges in deep neural networks and secondly applying deep learning models to some of the challenging problems in other domains, mainly in the areas of smartgrids and bioinformatics. All the work in this dissertation reflects my

contributions which have also been published in [61–64].

To improve training process of existing deep learning models, we propose a novel gradient amplification approach along with a training strategy which addresses the challenges discussed above. We suggest a unique training strategy which includes amplification during certain epochs along with normal training with no amplification. In this method, gradients are dynamically increased for some layers during back propagation so that significant gradient values are propagated to the initial layers. This process is repeated for a few epochs along with the normal training process with no gradient amplification for the other epochs. When neural networks are trained using this method, we observe that the testing/training accuracies of the models improve and achieve higher accuracies faster, even at higher learning rates, and therefore reduces the training time of these deep learning models. There are several improvements in this approach. We first use random selection of layers and perform detailed analysis on different combinations of layer types, number of layers and varied amplification factor. Detailed step-wise analysis of training strategies is performed to demonstrate the best strategy with different learning rates. Based on this information, we formulate measures using gradient fluctuations of layers and determine the layers to be amplified. Gradient amplification is also extended to neuron level amplification on simple deep learning models.

Deep learning methods are now used in multiple domains to solve address some of the challenging problems. In this work, we apply deep learning in smart grids to detect data integrity attacks and in bioinformatics to classify the stage of Hepatitis C virus (HCV) infections. There have been attempts to add deep learning methods with power grid architectures to detect such attacks. In our work, an independent deep learning prediction model is designed based on the past secured and attacked measurements. In bioinformatics, we develop a image based preprocessing method to classify the stage of HCV infection. This is a crucial task as as it has an affinity to lead towards chronic infection with time due to its highly mutable nature. To our knowledge, there are no reliable diagnostic assays for distinguishing acute and chronic HCV infections. Although some machine learning models are known to work well for sequence data for classification problems, their straightforward application to

viral genomic data is problematic, since the number of viral sequences and the structures of intra-host viral populations are not consistent across various samples. We propose a novel preprocessing approach to transform irregular viral genomic data into a normalized image data. Such representation allows to apply powerful machine learning algorithms to the problem of classification of acute and chronic HCV infections. We then apply this image data to classify HCV infection as well as to detect outbreaks.

## 1.4 Outline

Chapters 2 introduces random gradient amplification method with detailed experiment results. Chapters 3 and 4 present methods and results of intelligent amplification applied at layer and neuron level respectively. Chapters 5 and 6 show applications on deep learning models on smart grid and bioinformatics areas respectively, with future works and conclusions in chapter 7.

# 2

# GRADIENT AMPLIFICATION BY RANDOM SELECTION

In this chapter, we propose a gradient amplification approach along with training steps which addresses the challenges mentioned earlier. In this method, gradients are dynamically increased for some layers(selected randomly) during backpropagation so that significant gradient values are propagated to the initial layers. This process is repeated for a few epochs along with the normal training process with no gradient amplification for the other epochs. When neural networks are trained using this method, we observe that the testing/training accuracies of the models improve and achieve higher accuracies faster, even with a higher learning rates, and therefore reduces the training time of these deep learning models. In the next section, we discuss our proposed method and also training strategy with a fixed learning rate schedule across epochs in detail. This chapter is based on the publication [61] and further details can be found therein.

## 2.1 Proposed Method

Our proposed approach is to dynamically amplify (increase) the value of the gradients for a selection of layers during backpropagation. This ensures that the gradient values are not diminished while updating weights for the initial network layers and a significant value of the gradients is available during backpropagation even for deep neural networks with large number of layers. This also accelerates the training time by making relatively larger weight updates. Layers are amplified randomly without any dependency on training data. Architectures of neural networks have evolved over the years and there are many different layers where such an amplification can be done. The layers on which gradient amplification can be performed during backpropagation are arranged into a group, say $G$. To determine this group, firstly, the type of layers that needs to be included for gradient amplification

should be identified. Each of the layers such as convolution layers, batch normalization layers, pooling layers, activation function layers and so on can be chosen to be included in the group. The type of the layer considered plays a crucial role in the performance of the model. Gradient amplification is done on a subset of the layers from this group $G$, which we refer as *amp* layers in the rest of the paper. Selection of the *amp* layers from a group of layers can be done in various methods. In this work, we determine the *amp* layers by random selection. To identify which subset size has better performance, we choose a parameter $\beta$ representing the ratio of *amp* layers to be selected from all the layers in the group $G$. Gradients are amplified when they pass through these randomly selected layers during backpropagation. During amplification, value of gradients is increased at run time by multiplying the actual gradient values by a factor $\Gamma$. The value of $\Gamma$ is important as it should not be too small or too large. If the value of $\Gamma$ is too small, then the increase might not be effective and if it is too large it might overfit the data or cause incorrect weight updates. During training, we perform gradient amplification for some epochs and with no gradient amplification for other epochs. Algorithm 1 describes the training process with gradient amplification and algorithm 2 describes the steps for the selection of layers from $G$.

There are mainly three important parameters while applying gradient amplification method namely, the type of the layers to be employed for amplification, the ratio of layers ($\beta$) to be chosen from selected layers to perform amplification and gradient amplification factor.

Here we perform three phase analysis while evaluating our model.

**Phase1** In this phase, we choose the type of layers to be considered for amplification. There are several types of layers at which amplification can be applied such as activation function layers, pooling layers, batch normalization layers and convolution layers. Convolution layers apply kernel functions and extract important features from the data and pooling layers perform accumulation of features over a grid using several strategies such as retrieving maximum values, minimum values, averaging, fractional pooling and so on. Since

**Algorithm 1** Training process with gradient amplification

**Input:** $M$, $params$=[$(e_1, \eta_1, \beta_1, \Gamma_1)$, $(e_2, \eta_2, \beta_2, \Gamma_2)$, ...]
*Variables:*
$\Gamma$ is gradient amplification factor
$\beta$ is ratio of layers to be selected for amplification
$\eta$ is the learning rate
*amp* the set of layers selected to perform amplification
$M$ is the neural network model
*params* is an array of elements, each in the format (*end_epoch*, $\eta$, $\beta$, $\Gamma$)

*start_epoch*=0
**for** $(e_i, \eta_i, \beta_i, \Gamma_i)$ in *params* **do**
  *update learning rate to* $\eta_i$
  optimizer=sdg_optimizer($\eta_i$)
  **if** $(\beta_i > 0)$ **then**
    $amp$ = GetGradientAmpLayers($M$, $\beta$ )
  **end if**
  **for** $k$ = *start_epoch* to $e_i$ **do**
    train the model $M$
    **if** $(\beta_i > 0)$ **then**
      multiply gradients with $\Gamma_i$ for layers in *amp* during backpropagation
    **else**
      perform regular backpropagation without gradient amplification
    **end if**
  **end for**
  *start_epoch*=$e_i$
  reset *amp*
  evaluate model $M$ with a testing set
**end for**
**return**

**Algorithm 2** Determination of *amp* layers

**Input:** in $M$, $\beta$
$\beta$ is ratio of layers to be selected for amplification
$G$ is a set consisting of a group of all layers that can be used for gradient amplification
*layer_types*= Set indicating the type of layers to be used for amplification
Function GetGradientAmpLayers($M$, $\beta$ )
**for all** *layer* in *layer_types* **do**
  include *layer* in $G$
**end for**
*amp_size* = $\beta$*sizeof($G$)
$amp$ = RandomSelect($G$, *amp_size*)
EndFunction

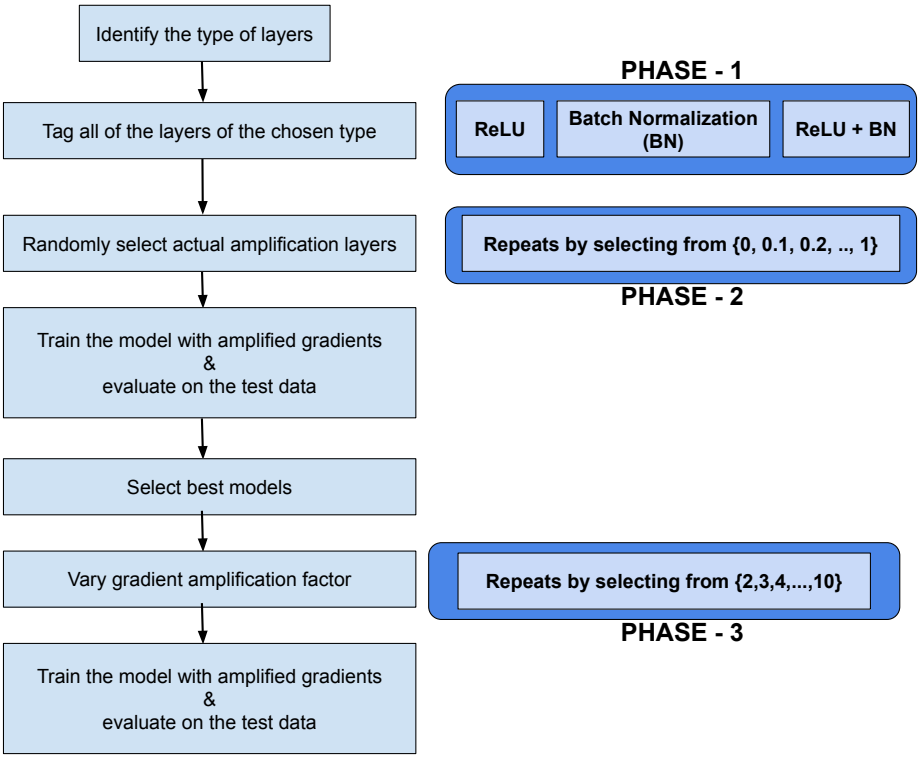



Figure (2.1) Overview of all the experiments performed by varying different parameters of gradient amplification.

the network parameter tuning while training can be sensitive to these values, in this work, we do not perform amplification on these layers. Batch normalization layers normalize data over a batch of inputs, and activation function layers transform data non-linearly before forwarding it to the succeeding layers. In our work, we perform gradient amplification on batch normalization and activation function layers. ReLU is the activation function used in Resnet and VGG models. From these two types of layers, either one or both of them can be considered for amplification. Once the type of the layers is selected, we now tag all the layers of the selected type to belong to the group $G$. We now move to the next phase to determine the final amplification layers *amp*.

**Phase2** Once the set of layers $G$ is determined, the next task is to find the subset of layers which gives better performance. It requires identifying subset size and selection of those many layers from $G$. Since the size is unknown, experiments are performed by selecting the size to be a ratio of size of $G$. This ratio. $\beta$, is chosen from the set $\beta \in \{0, 0.1, 0.2, ..., 0.9, 1\}$. The actual size of *amp* is determined by the value $\beta \times size\_of(G)$. When the value is 0, no layers are chosen and gradient amplification is not performed. When the value is 1, then all the layers in $G$ are considered for amplification. 0 is included to verify whether the model performs better without gradient amplification or vice versa. Random selection is employed to select *amp* subset of layers from $G$. We perform experiments with all these sizes and select the model with the best performance.

**Phase3** In this phase, the layers *amp* on which gradient amplification can be applied are known. The only parameter left to explore is $\Gamma$, the factor with which gradient needs to be amplified. To reduce computation complexity in testing all the combinations of parameter values *amp*, $\beta$ and $\Gamma$, firstly experiments are performed on all combinations of *amp* and $\beta$ i.e., until phase-2, then the best models are chosen from phase-2 and analyzed by varying $\Gamma$. The value of $\Gamma$ is firstly varied from $\{1, 2, 3, .., 10\}$ to analyze the impact of amplification and then fine-tuned by varying from $\{1.1, 1.2, ..., 2.9, 3.0\}$ to determine the value that works best during training.

## 2.2 Experiments & Results

### 2.2.1 Setup

Our experiments are performed on CIFAR-10 dataset [65] which consists of 60000 colored images of 10 classes with 6000 images per class and each image has 32x32 resolution. We implement our algorithms using python and pytorch [66] libraries. In our experiments, we employ several standard deep learning models and train them for 150 epochs. The number of epochs, combination of number of epochs and learning rates can be chosen as one thinks best. In this work, the first 100 epochs have learning rate of 0.1 and the next 50 epochs have the learning rate of 0.01 (as shown in Fig. 2.2. The first 50 epochs are trained with learning rate of 0.1 without gradient amplification. This is because for the first few epochs, the model is considered to be in transient phase and the network parameters undergo significant changes. This initial transient can be considered for any number of epochs and in this work, we set it to 50 epochs. The next 50 epochs have the same learning rate of 0.1 but has gradient amplification applied during backpropagation while training the model (as shown in Fig. 2.3(a)). After identifying the best *params* with gradient amplification for epochs $51 - 100$, using those *params* for those epochs, we extend amplification for epochs $101 - 130$ to identify the best *params* and with no amplification for epochs $131 - 150$, as shown Fig. 2.3(b). There are mainly three important parameters while applying gradient amplification method namely, the type of the layers to be employed for amplification, the ratio of layers ($\beta$) to be chosen from selected layers to perform amplification and gradient amplification factor. The effects of varying each of these parameters are explained in detail in the subsections below. We run our experiments on Resnet and VGG models with different architectures.

### 2.2.2 Results

In our experiments, we employ Resnet-18, Resnet-34 and VGG-19 models and perform thorough analysis. As the complexity of the model and the depth of the network increases,

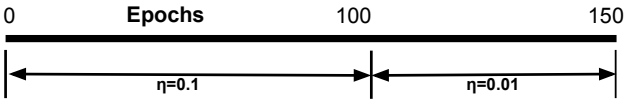



Figure (2.2) Experiment setting showing the number of epochs and learning rates corresponding to epochs for training all the models.

it takes longer to compute and requires more GPU/CPU resources. Since we perform many experiments (around hundreds), having models with relatively simpler architectures and less layers would make the computation time faster.

While performing experiments, we choose either batch normalization layers or ReLU layers or both and then verify their performance over multiple epochs. We first explain the training *params* which is important to understand the performance tables. We train our models for 150 epochs and the learning rate of the first 100 epochs is 0.1 and the next 50 is 0.01. We train the models with no gradient amplification for the first 50 epochs as the initial transient and for the next epochs, we aim to identify the pattern to select the epochs which improve the overall performance of the model. We follow the training steps mentioned in Algorithm 1 and $params$=$[(e_1, \eta_1, \beta_1, \Gamma_1)$, $(e_2, \eta_2, \beta_2, \Gamma_2)$, ...] is chosen as $[(50, 0.1, 0, 1), (100, 0.1, 0, 1), (130, 0.01, 0, 1), (150, 0.01, 0,$

$1)]$ when no gradient amplification is performed. The values in each element represent end epoch, learning rate, ratio of amplified layers and gradient amplification factor respectively. For instance, $(50, 0.1, 0, 1)$ means that the model is trained with learning rate 0.1 until we reach 50 epochs, during which 0 layers are selected for gradient amplification and amplification factor is 1.

Performance of original models with no gradient amplification is firstly recorded. Next, models with gradient amplification are experimented in two steps. We first set *params* as $[(50, 0.1, 0, 1), (100, 0.1,xx, 2), (130, 0.01, 0, 1), (150, 0.01, 0,$

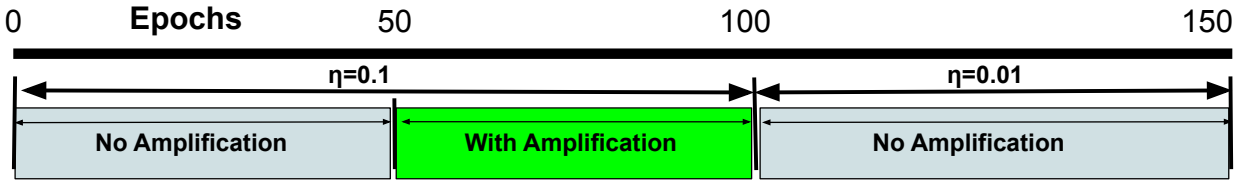
$1)]$. That is, no gradient amplification is applied for the first and the last 50 epochs, as shown in Fig. 2.3(a). For epochs $51 - 100$, the ratio of selected layers is scanned from $\{0, 0.1, 0.2, ..., 1\}$ to identify the best model with the amplification factor 2. For simplicity,

we define $S1\_\{mm\}$ to represent the modified ratio $mm$ during epochs 51-100 in step-1 while performing amplification, and $S2\_\{mm\}\_\{nn\}$ to represent the modified ratio $mm$ during epochs 51-100 and $nn$ during epochs 101-130, respectively, during amplification in step-2. So, the *params* defined above will be represented as $S1\_xx$, where $xx$ represents the value that is varied. Once we identify the best ratio for $51 - 100$ epochs, say 0.7, we then run the experiments with different ratio values for the next 30 epochs by setting *params* to be $S2\_0.7\_xx$ i.e., $[(50, 0.1, 0, 1), (100, 0.1, 0.7, 2), (130, 0.01, xx, 2), (150, 0.01, 0, 1)]$ as shown in Fig. 2.3(b). Note that, the learning rate is decreased to 0.01 after 100 epochs. After these experiments, the best models are chosen to perform experiments in phase-3 to analyze the impact of gradient amplification factor on its performance. All the phases and various experiments performed are shown in Fig. 2.1.

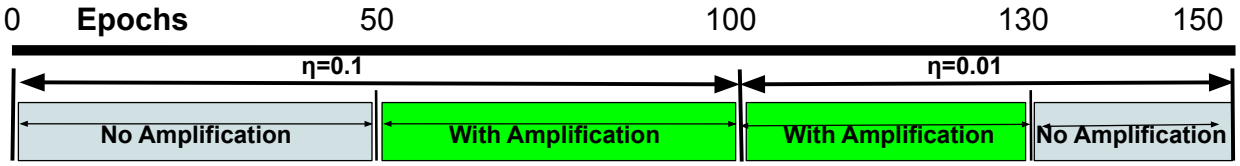
From our initial experiments, we observe that the ratio values $\{0.1, 0.3, 0.5, 0.6\}$ on average provide better results in step-1, explained below in detail in Phase-2. Instead of running step-2 only on the best models from step-1, different models are built with ratio values $\{0.1, 0.3, 0.5, 0.6\}$ for epochs 51-100 where the learning rate is 0.1. We perform our analysis on phase-1 and phase-2 for the following amplification *params* in step-2 (see 2.3(b)) :

- $S2\_0.1\_xx$: $[(100, 0.1, \mathit{0.1}, 2), (130, 0.01, xx, 2)]$
- $S2\_0.3\_xx$: $[(100, 0.1, \mathit{0.3}, 2), (130, 0.01, xx, 2)]$
- $S2\_0.5\_xx$: $[(100, 0.1, \mathit{0.5}, 2), (130, 0.01, xx, 2)]$
- $S2\_0.6\_xx$: $[(100, 0.1, \mathit{0.6}, 2), (130, 0.01, xx, 2)]$

**Phase-1: (Effect of type of layers)** In this work, ReLU, BN or both are the layers used for gradient amplification. We run original models without gradient amplification 5 times and record their training, testing accuracies and compare the corresponding gradient amplified models with the mean of the these accuracies across 5 runs. For each type(s) of layer chosen, experiments are run for *params* $S2\_0.1\_xx$, $S2\_0.3\_xx$, $S2\_0.5\_xx$, $S2\_0.6\_xx$




(a) Step-1

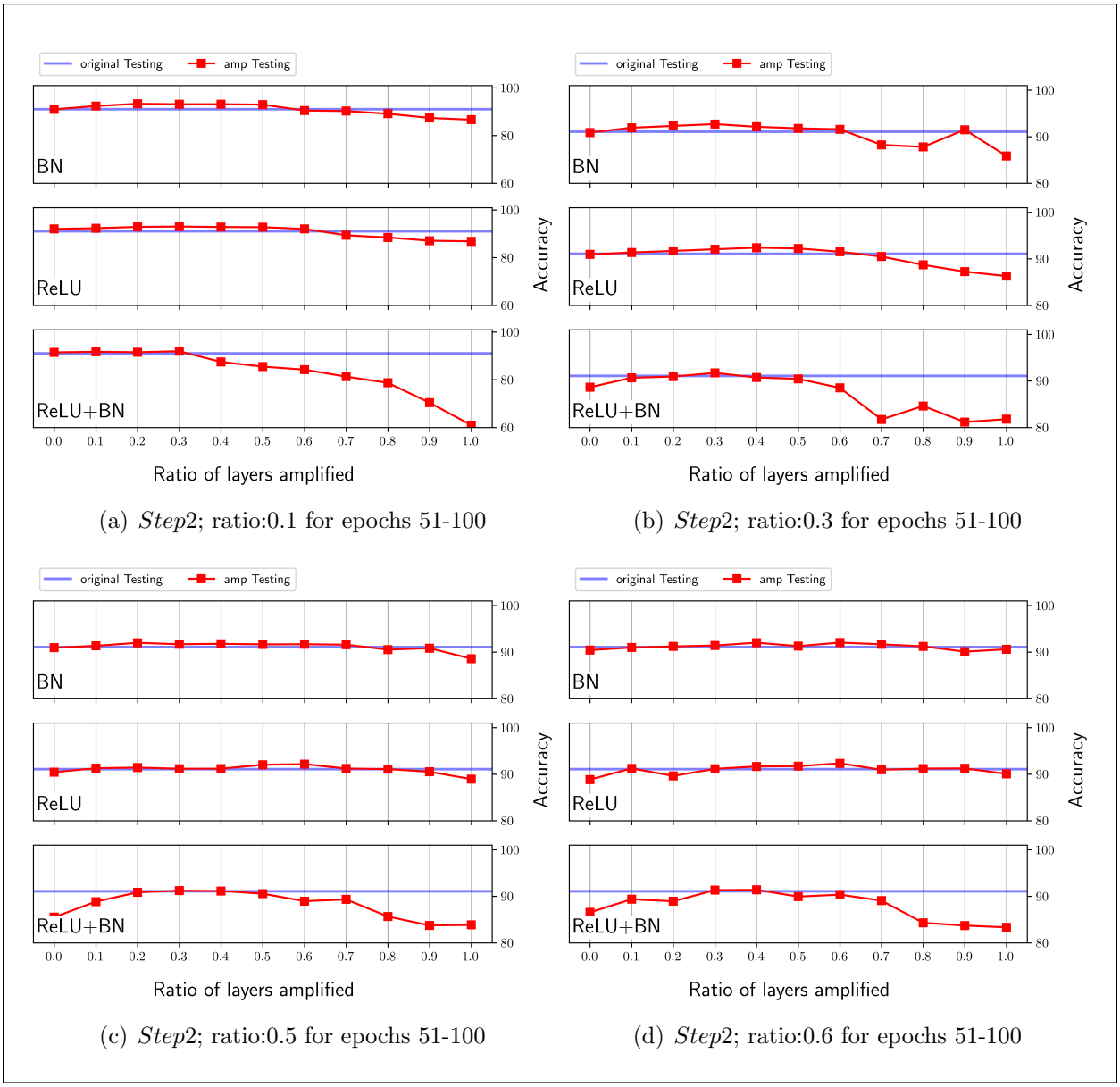



(b) Step-2

Figure (2.3) Two step training process carried out during performance analysis of deep learning models. Experiments are first executed on the models with training steps shown in step-1 (a). For step-2(b), ratio parameters for gradient amplification which have better performance of the models in step-1 are considered as the parameters for epochs 51-100 epochs and experiments are performed by analyzing ratio parameters for epochs 101-130, with no amplification from epochs 131-150. These settings show the number of epochs and the learning rates corresponding to these epochs while training these models.

and the best training and testing accuracies of these models are compared with the average training and testing accuracies of corresponding original model. Since training accuracies of the original models are close to 100%, we emphasize our comparison on testing accuracies.

In VGG-19 model, we perform analysis considering ReLU, BN or both layers for gradient amplification and provide accuracy improvements for *params* $S2\_0.1\_xx$, $S2\_0.3\_xx$, $S2\_0.5\_xx$, $S2\_0.6\_xx$ respectively. When only ReLU layers are chosen, testing accuracies improve around 1.98%, 1.31%, 1.08%, 1.25% respectively for above *params*. In the case of amplification applied only to BN layers, an improvement of 2.27%, 1.64%, 0.91%, 0.96% in testing accuracies is observed. When both ReLU and BN are chosen, models have accuracy difference of 0.9%, 0.64%, 0.13%, 0.3% respectively. When both layers are used in amplification, the improvements across different models are less than 1%, which becomes better when only either ReLU or BN is used. Best improvements are seen when amplification is applied on only BN layers. Fig. 2.4 shows the performance of VGG-19 models for all the above *params* with different types of layers.



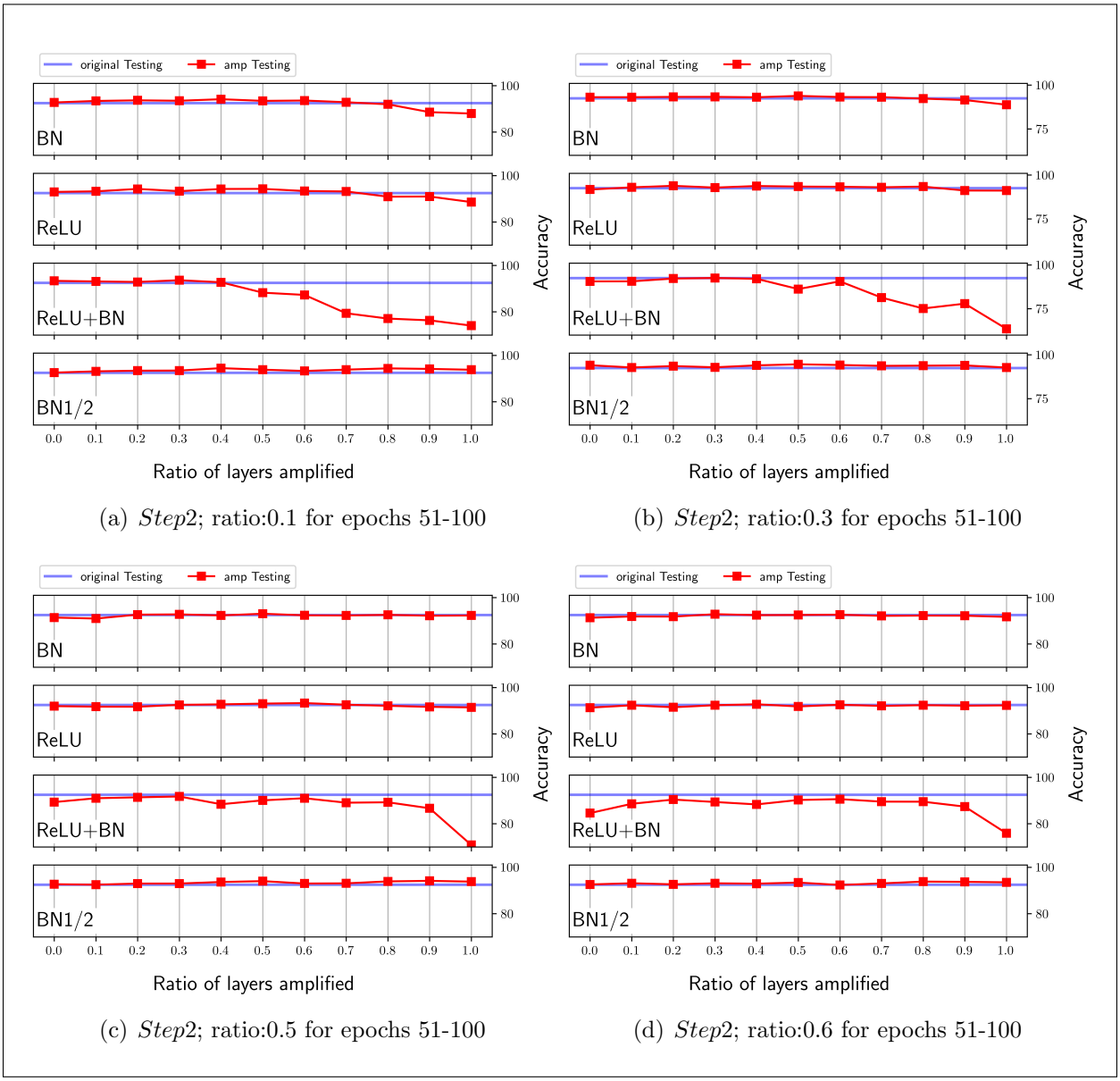

(a) *Step*2; ratio:0.1 for epochs 51-100

(b) *Step*2; ratio:0.3 for epochs 51-100

(c) *Step*2; ratio:0.5 for epochs 51-100

(d) *Step*2; ratio:0.6 for epochs 51-100

Figure (2.4) Performance of VGG 19 models for various amplification *params* are shown. In each plot, blue horizontal line shows the average testing accuracy of the original models without gradient amplification. amp testing refers to testing accuracies of models with gradient amplification. The type of the layer is shown in each subplot; horizontal and vertical axes correspond to the ratio of amplified layers and accuracies respectively.



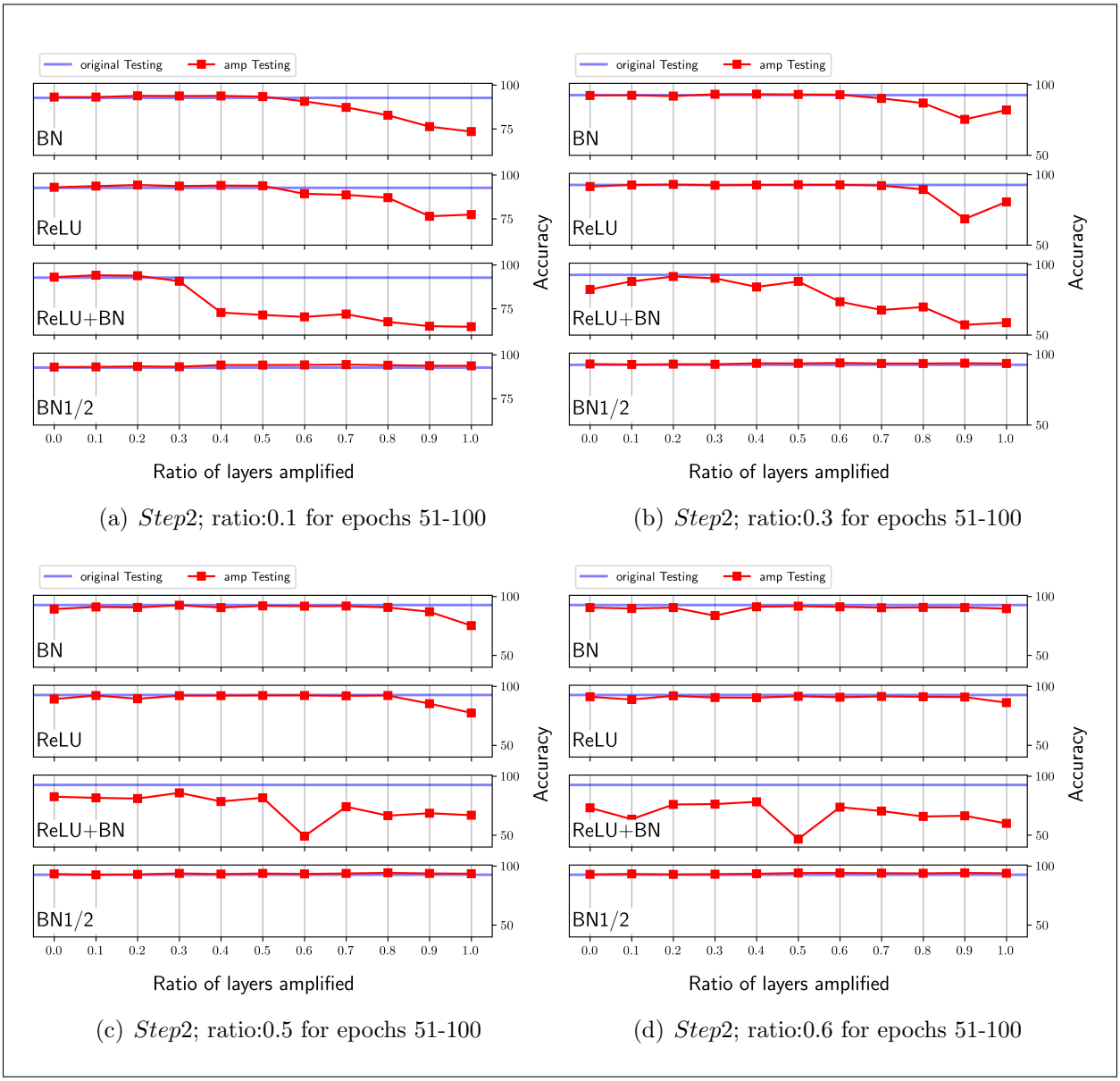

(a) *Step*2; ratio:0.1 for epochs 51-100

(b) *Step*2; ratio:0.3 for epochs 51-100

(c) *Step*2; ratio:0.5 for epochs 51-100

(d) *Step*2; ratio:0.6 for epochs 51-100

Figure (2.5) Performance of Resnet-18 models for various amplification *params* (red) compared to mean accuracies of the original models(blue) with no gradient amplification.)



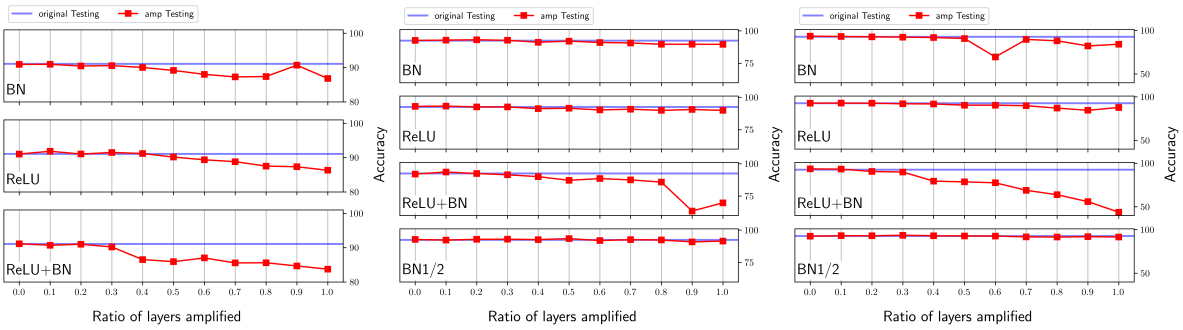

(a) *Step*2; ratio:0.1 for epochs 51-100

(b) *Step*2; ratio:0.3 for epochs 51-100

(c) *Step*2; ratio:0.5 for epochs 51-100

(d) *Step*2; ratio:0.6 for epochs 51-100

Figure (2.6) Performance of Resnet-34 models for various amplification *params* (red) compared to mean accuracies of the original models(blue) with no gradient amplification.)




(a) *Step*1 for $VGG-19$ (b) *Step*1 for $Resnet-18$ (c) *Step*1 for $Resnet-34$

Figure (2.7) Performance of the models after training with step-1 strategy with gradient amplification(red) applied from epochs 51-100 compared to mean accuracies of the original models(blue) with no gradient amplification.

Resnet models are made of residual blocks, each of which consists of two convolution units and therefore each block has two ReLU and BN layers. In these models, other than experimenting with all ReLU and BN layers, we additionally perform experiments considering only one of the BN layers from residual blocks. When all the BN layers are considered for amplification in Resnet-18 models, there is an improvement of 1.66%, 1.35%, 0.53%, 0.34% respectively for *params* $S2\_0.1\_xx$, $S2\_0.3\_xx$, $S2\_0.5\_xx$, $S2\_0.6\_xx$. When only ReLU layers are used, there is a difference of 1.76%, 1.32%, 0.77%, 0.26% respectively. When both BN and ReLU are used, there is an initial improvement of 01.14%, 0.03%, for *params* $S2\_0.1\_xx$, $S2\_0.3\_xx$ and then accuracies drop to $-0.738\%$, $-1.88\%$ respectively for *params* , $S2\_0.5\_xx$, $S2\_0.6\_xx$. When only one of the BN layer from a residual block is considered, there is an improvement of 1.98%, 2.08%, 1.64%, 1.32% for respective *params*. When both BN and ReLU are used for amplification, there is an improvement only for params $S2\_0.1\_xx$ and either declines or slightly changes for the remaining *params*. When either ReLU or BN layers is considered, for params $S2\_0.1\_xx$, $S2\_0.3\_xx$, more than performance improvement of more than 1.3% can be observed and for params $S2\_0.5\_xx$, $S2\_0.6\_xx$, improvements are less than 1%. When one of the BN layers in residual blocks are considered, then the models have accuracy improvements of more than 1.3% for all params and it also achieves the best

testing accuracy of 94.57% with an improvement of 2.08% over original model. Fig. 2.5 shows the performance of various Resnet-18 models for all the above *params* with different types of layers.

Similarly for Resnet-34, in the case of only BN layers, there is an accuracy gain of 1.21%, 0.64% for *params* $S2\_0.1\_xx$, $S2\_0.3\_xx$ and then a drop of $-0.15\%$, $-0.94\%$ for *params* $S2\_0.5\_xx$, $S2\_0.6\_xx$ respectively. When only ReLU layers are considered, there is an improvement of 1.61%, 0.42% for $S2\_0.1\_xx$, $S2\_0.3\_xx$, and for other *params*, there is a decrease in accuracy of $-0.43\%$, $-0.78\%$ respectively for *params* $S2\_0.5\_xx$, $S2\_0.6\_xx$. When both BN and ReLU are used, there is an initial improvement of 1.14% for *params* $S2\_0.1\_xx$ and then accuracies drop to $-1.19\%$, $-6.78\%$, $-14.36\%$ respectively for *params* $S2\_0.3\_xx$, $S2\_0.5\_xx$, $S2\_0.6\_xx$. When only one of the BN layers are used in a residual block, an improvement of 1.67%, 1.23%, 1.55%, 1.49% can be seen for respective *params*. When all the BN layers are only used for amplification, there is an improvement of more than 1% only for params $S2\_0.1\_xx$ and the performance either declines or slightly changes for remaining *params*. Similar pattern is observed when only ReLU or both ReLU+BN are used for amplification. When one of the BN layers in residual blocks are considered, models have accuracy improvements of more than 1.2% for all params and it also achieves the best testing accuracy of 94.39% with an improvement of 1.67% over original model. Fig. 2.6 shows the performance of Resnet-34 models for all the above *params* with different types of layers.

**Phase-2: (Effect of ratio of selected layers $\beta$)** Here, we discuss the impact of the ratio of selected layers on each of the above types. In our training strategy, gradient amplification is firstly applied in step-1 (as shown in Fig. 2.3(a)) to determine the best performing ratio values for epochs 51-100. Fig. 2.7 shows the performance of different models with different type of layers selected for amplification. The best training and testing accuracies after gradient amplification across all the ratio values are compared with the original baseline models to analyze the overall effectiveness. In VGG-19, for all layer types, as the ratio

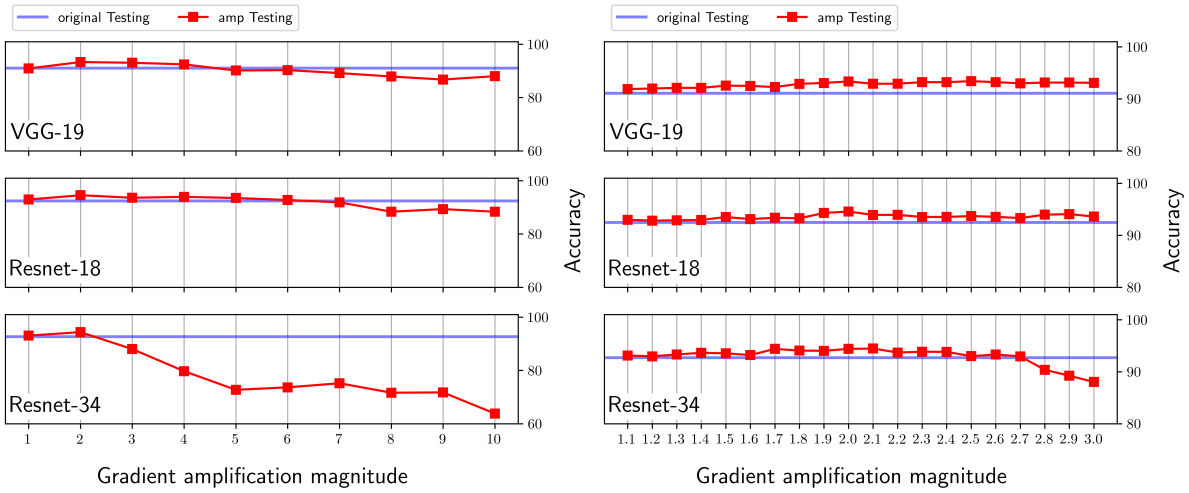
of amplified layers increases, the performance of the model diminishes compared to original models. The training accuracies decrease at an increased rate compared to testing accuracies. When gradient amplification is performed, training and testing accuracies decrease slightly by $-0.3\%$ and $-0.14\%$ (for BN only) and increase slightly by 0.02% and 0.03%(in the case of ReLU+BN) and show an improvement of 0.65% and 0.77% (for ReLU only) respectively. In Resnet-18 and Resnet-34 models, as the ratio of amplified layers increases, training and testing accuracies remain close to the accuracies of the baseline models when only one of the BN layers in a residual block are considered. When either BN or ReLU is considered, as the ratio of amplified layers increases, performance of the models decreases slightly compared to original models. When both BN and ReLU are considered for amplification, as the ratio of layers increases, performance of the models decrease significantly compared to respective baseline models. In Resnet-18, the best training and testing accuracies after gradient amplification across all the ratio values, show an improvement by 0.69% and 0.62% (for BN only), 0.52% and 0.67% (for ReLU only), 0.52% and 0.67%(in the case of ReLU+BN), and 0.79% and 0.92% (in the case when one of BN layers from residual block) respectively. In Resnet-34, the best training and testing accuracies after gradient amplification across all the ratio values, show an improvement by 0.54% and 0.68% (for BN only), 0.4% and 0.15% (for ReLU only), 0.42% and 0.81%(in the case of ReLU+BN), and 0.57% and 0.85% (when one of the BN layers from residual blocks are considered) respectively.

In the case of step-1, amplification is applied only for epochs 51-100 ($\eta = 0.1$). We also perform experiments by applying amplification for epochs 101-150 ($\eta = 0.01$), by considering all or some of the epochs. We observe that the models perform better when amplification is applied from 51-100 epochs ($\eta = 0.1$) followed by 101-130 epochs ($\eta = 0.01$) as shown in step-2 (Fig. 2.3(b)). To narrow the parameter space, we only consider ratio values for epochs 51-100 where the models perform better. From analysis of model performances in step-1, ratio values $\{0.1, 0.3, 0.5, 0.6\}$ on average provide better results and therefore, these ratios are used for epochs 51-100 ($\eta = 0.1$) as mentioned earlier and ratio values are varied for 101-130 epochs ($\eta = 0.01$), namely $S2\_0.1\_xx$, $S2\_0.3\_xx$, $S2\_0.5\_xx$, $S2\_0.6\_xx$. Fig. 2.4, 2.5 and 2.6

show the performance of VGG-19, Resnet-18 and Resnet-34 models respectively for these params when different layers are amplified. Performance improvements of these models are discussed in Phase-1 in detail. Here we emphasize on the effect on the models as the ratio of amplified layers increases. For VGG-19 models, as the ratio of layers increases, there is an increase in performance initially and then it decreases when the ratios above 0.7. When both ReLU+BN are amplified, the models have significant decrease with the increase of ratio values. In the case of Resnet-18, models have improved or similar performance even as the ratio increases except when both ReLU+BN are amplified, in which case it decreases. For Resnet-34, models have improved or similar performance even as the ratio increases until 0.8, after which it decreases. But when both ReLU+BN are amplified, the performance declines even for smaller ratios.

When amplification is applied using approach in step-1 (Fig. 2.3(a)), the models perform better when only ReLU are amplified in the case of VGG-19 and for Resnet-18, Resnet-34, models perform best when only one of the BN layers from a residual block is used for amplification. When amplification is done as in step-2, all models achieve higher accuracies than baseline models for most of the ratio values except when ReLU+BN are amplified, in which case only some of the smaller ratio values have better models. This shows that a small ratio of amplified layers are sufficient to improve the performance of original models.

**Phase-3: (Effect of gradient amplification factor)** Fig. 2.8(a) shows the performance of models as $\Gamma$ is varied. *params* of the best models after analysis phase-1 and phase-2 are taken and gradients are amplified by varying the value of $\Gamma$ from $\{1, 2, 3, .., 10\}$. For VGG-19, the best model is achieved while amplifying only BN layers for *params* $S2\_0.1\_0.3$. For Resnet-18 and Resnet-34, the best models are achieved while amplifying only one of the BN layers in residual units and for *params* $S2\_0.3\_0.5$ and $S2\_0.1\_0.7$ respectively. While changing values of $\Gamma$, as the factor of amplification $\Gamma$ increases, the performance of the models declines. To generalize, we can say that when $\Gamma$ is more than 5, the models do not perform better or sometimes perform worse than the corresponding baseline models. Effect of ampli-




(a) Larger amplification factor steps

(b) Fine tuned amplification factor steps

Figure (2.8) Performance comparison of amplified models(red) as $\Gamma$ is varied from 1 to 10 (horizontal axis) (vs) original models (blue). Right plots correspond to comparison of testing accuracies of amplified models(red) as $\Gamma$ is varied in small steps from 1 to 3 (horizontal axis) (vs) original models(blue).

fication factor $\Gamma$ also depends on the ratio of layers being amplified. If the ratio is close to 1, then $\Gamma$ values less than 5 can also decrease performance of the models. For instance, one such example is shown in Fig. 2.9 where the gradients are modified for Resnet-34 model for *params* $S2\_0.3\_0.9$, in which only one of the BN layers of residual units are being amplified. This model has larger ratio values with testing accuracy of 94.21% which is close to the best Resnet-34 model shown in 2.1. In this case, the models perform better for $\Gamma \in \{1, 2\}$ and performance of the model decreases for values greater or equal to 3. It can also be seen that all the models have better accuracies for $\Gamma = 2$ even for different ratio of amplified layers which justifies all our analysis done in phase-1 and phase-2.

We also perform experiments by fine-tuning the amplification factor from 1 to 3 in steps of 0.1, i.e., by varying $\Gamma \in \{1.1, 1.2.1.3, 1.4, ...., 3\}$. Fig. 2.8(b) shows the performance of these models as $\Gamma$ is varied in small steps from 1 to 3. In the case of VGG-19 and Resnet-18, the model always performs better than the baseline models both during training and testing and for Resnet-34, the model performs better until 2.7 and declines after that. In all these

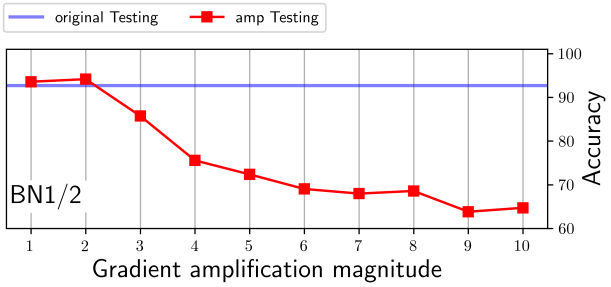
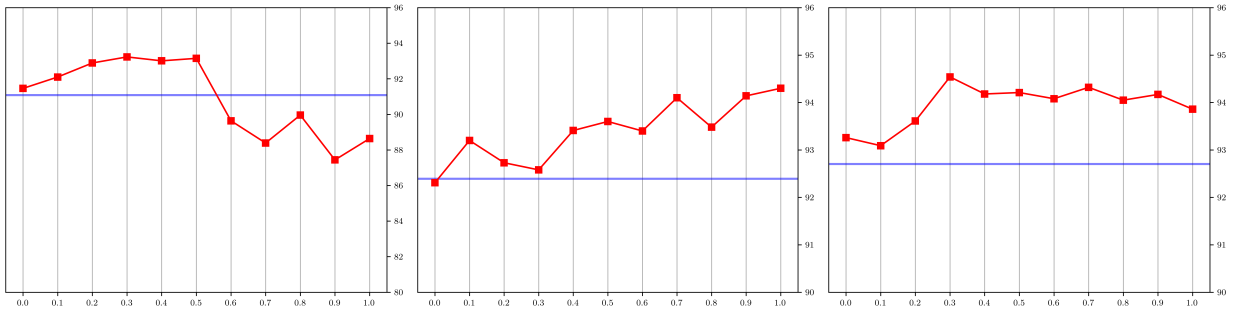



Figure (2.9) Performance of Resnet-34 model with amplification(red) for *params* $S2\_0.3\_0.9$ when $\Gamma$ is varied from 1 to 10 (horizontal axis) (vs) original models.
Performance of the models degrade significantly as $\Gamma$ value crosses 3.

models, it can be observed that the best accuracy is around the value 2 which justifies our experiment analysis in the above phases.

**Applying random amplification** Here we apply random amplification factor in the range of [1.75, 2.25] on the best models. In this step, every time the layers are selected to perform amplification, a random value is chosen in range of [1.75, 2.25]. Experiments are performed on VGG-19 with *params* $S2\_0.1\_xx$ while using batch normalization layers$xx$ for amplification. In resnet-18 and resnet-34, only one of the batch normalization layers is considered with *params* $S2\_0.2\_xx$ and *params* $S2\_0.1\_xx$ respectively. Fig. 2.10 shows the performance of these models as the ratio of layers are modified from 0 to 1 in steps of 0.1. It can be seen that for VGG-19 models, performance of the model increases till it reaches 0.5 reaching a maximum testing accuracy of 93.23% and then decreases after that. In the case of resnet-18, accuracies increase with the increase of ratio of selected layers for amplification reaching a maximum testing accuracy of 94.3%. In resnet-34, amplified models have better performance for all ratio values, accuracies increase until it reaches the ratio of 0.3 and then decrease very slowly reaching the maximum testing accuracy of 94.540%. It can be observed that randomly selecting amplification factor has similar performance compared to using a constant amplification factor of 2.



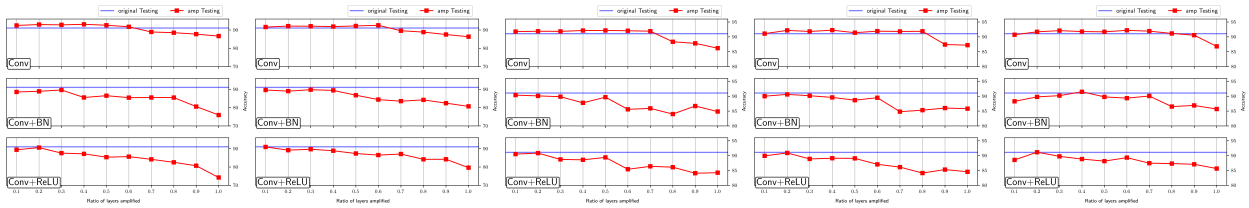
(a) VGG-19 (b) Resnet-18 (c) Resnet-34

Figure (2.10) Performance of the best models with gradient amplification over 150 epochs compared to original model with no gradient amplification. Original training(gray), testing(blue) accuracies including their mean accuracies are plotted along with amplified training(green) and testing(red) accuracies. These plots demonstrate that the models do not overfit while training with amplification.

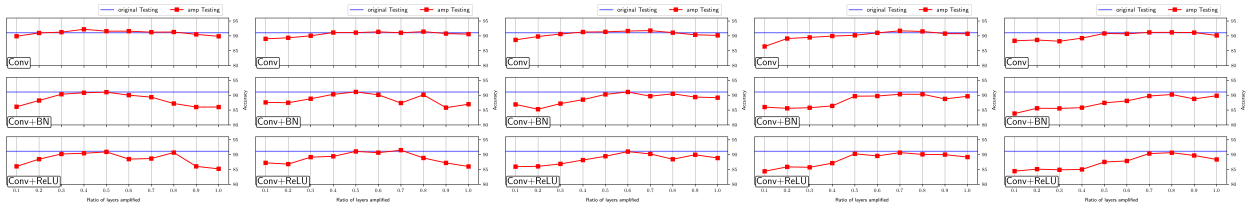
### 2.2.3 Performing amplification including convolution layers

In this section, we perform detailed experiments by also considering convolution layers for amplification. Fig. 2.11, 2.12, 2.13 show the performance of various models with different typed of layers combined with convolution layers. units are selected. It can be observed that models have better performance when only convolution layers modified and has best performance for resnet models when only of the convolution layers is amplified. In this section, we also perform detailed analysis by analyzing performances of all the combinations of types of layers as well as different *params*. Performance of the model declines when ReLU or BN layers are also selected along with convolution layers. The decline is significant with the increase in the ratio of selected layers. This again shows that amplifying a few layers is sufficient to obtain performance improvement. Another important aspect in *params* is to allow a last few epochs to perform updates without amplification so that the models can converge to better solutions.

CIFAR-10 dataset, VGG-19 model



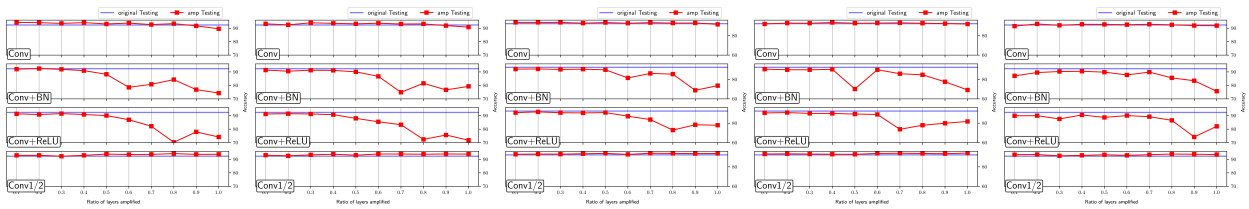
(a) *Step*2; ratio:0.1($\eta = 0.1$) (b) *Step*2; ratio:0.2($\eta = 0.1$) (c) *Step*2; ratio:0.3($\eta = 0.1$) (d) *Step*2; ratio:0.4($\eta = 0.1$) (e) *Step*2; ratio:0.5($\eta = 0.1$)



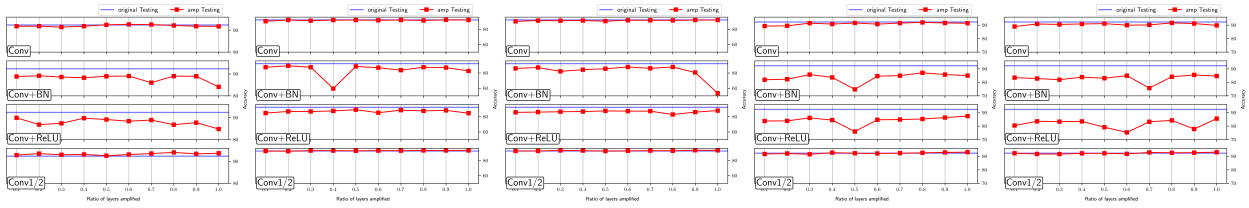
(f) *Step*2; ratio:0.6($\eta = 0.1$) (g) *Step*2; ratio:0.7($\eta = 0.1$) (h) *Step*2; ratio:0.8($\eta = 0.1$) (i) *Step*2; ratio:0.9($\eta = 0.1$) (j) *Step*2; ratio:1($\eta = 0.1$)

Figure (2.11) Testing accuracies(Y-axis) of random amplification on VGG-19 model with increasing ratio of layers(X-axis) for learning rate($\eta = 0.01$ for different combinations of convolution, batch normalization and ReLU layers. Each of the subplot corresponds to a different ratio of layers selected for learning rate 0.1, for instance plot (a) refers to $S2\_0.1\_xx$, (b) refers to $S2\_0.2\_xx$ and so on.

CIFAR-10 dataset, Resnet-18 model



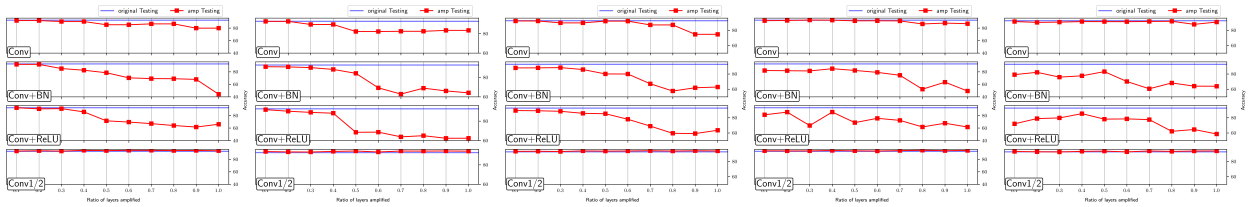
(a) *Step*2; ratio:0.1($\eta = 0.1$) (b) *Step*2; ratio:0.2($\eta = 0.1$) (c) *Step*2; ratio:0.3($\eta = 0.1$) (d) *Step*2; ratio:0.4($\eta = 0.1$) (e) *Step*2; ratio:0.5($\eta = 0.1$)



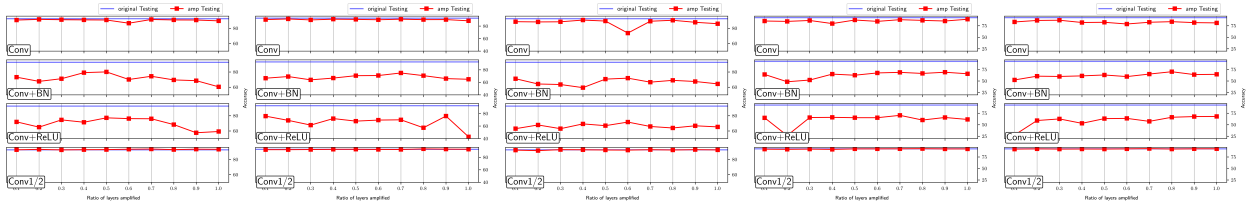
(f) *Step*2; ratio:0.6($\eta = 0.1$) (g) *Step*2; ratio:0.7($\eta = 0.1$) (h) *Step*2; ratio:0.8($\eta = 0.1$) (i) *Step*2; ratio:0.9($\eta = 0.1$) (j) *Step*2; ratio:1($\eta = 0.1$)

Figure (2.12) Testing accuracies(Y-axis) of random amplification on resnet-18 model with increasing ratio of layers(X-axis) for learning rate($\eta = 0.01$ for different combinations of convolution, batch normalization and ReLU layers.

CIFAR-10 dataset, Resnet-34 model




(a) *Step*2; ratio:0.1($\eta = 0.1$) (b) *Step*2; ratio:0.2($\eta = 0.1$) (c) *Step*2; ratio:0.3($\eta = 0.1$) (d) *Step*2; ratio:0.4($\eta = 0.1$) (e) *Step*2; ratio:0.5($\eta = 0.1$)




(f) *Step*2; ratio:0.6($\eta = 0.1$) (g) *Step*2; ratio:0.7($\eta = 0.1$) (h) *Step*2; ratio:0.8($\eta = 0.1$) (i) *Step*2; ratio:0.9($\eta = 0.1$) (j) *Step*2; ratio:1($\eta = 0.1$)

Figure (2.13) Testing accuracies(Y-axis) of random amplification on resnet-34 model with increasing ratio of layers(X-axis) for learning rate($\eta = 0.01$ for different combinations of convolution, batch normalization and ReLU layers.

### 2.2.4 Best models

The best performance of all the models is shown in Table 2.1. Performance improvements can be observed both in and testing accuracies. The rows with 'original' in *params* column show the performance of the original model with no gradient amplification. The following grayed row shows the performance of the corresponding model with gradient amplification. The *params* that achieve these best models are also shown. We can observe that gradient amplification increases both training and testing accuracies. Though training accuracies are very close to 100% in the original model, gradient amplification improves them further. It can be noted that resnet models comprises of residual blocks architecture (an extra connection to the preceding layer) which already overcomes vanishing gradients by passing the current gradients directly to the previous layers without modification using residual connection, and therefore an improvement of 1.67% can be assumed to be significant.

Fig. 2.14 shows the performance of the each of the models for the params listed in

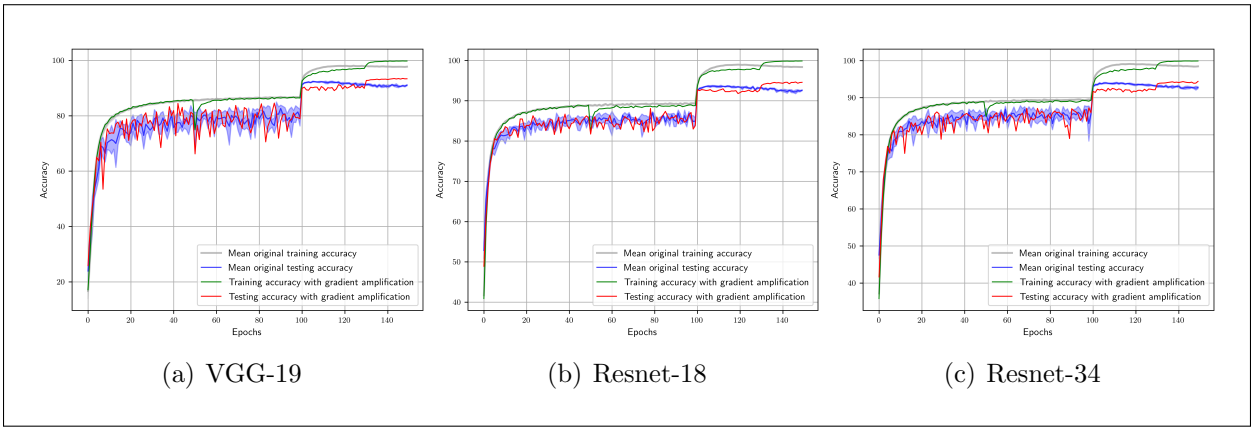
Table (2.1) Accuracy comparison of models with gradient amplification with random layer selection(vs) mean accuracies of corresponding original models across 5 runs.

| Model | *params* | Mean/Best accuracy (%) | | Improved accuracy (%) | |
|---|---|---|---|---|---|
| | | Training | Testing | Training | Testing |
| VGG_19 | Original | 97.87 | 91.08 | – | – |
| | *Ours* (*VGG_19* with amplification) | 99.764 | 93.35 | 1.9 | **2.27** |
| Resnet_18 | Original | 98.371 | 92.39 | – | – |
| | *Ours* (*Resnet_18* with amplification) | 99.878 | 94.57 | 1.51 | **2.18** |
| Resnet_34 | Original | 98.444 | 92.716 | – | – |
| | *Ours* (*Resnet_34* with amplification) | 99.774 | 94.39 | 1.25 | **1.67** |

Table 2.1. These plots demonstrate that the models trained with gradient amplification do not cause overfitting problem. In the case of VGG-19, the best model is achieved when amplification is applied only on BN layers and for resnet models, the best models are achieved when only one of the BN layers from a residual block are considered for amplification. Gradient amplification model surpasses the performance of all the original models. Accuracies achieved by amplified models not only exceed the mean average accuracies across 5 runs of the original models, but also outperform the best accuracy among these 5 runs of the original models.

## 2.3 Summary

In this work, we propose a novel gradient amplification method to dynamically increase gradients during backpropagation. We also provide a training strategy consisting of set of epochs with switching between gradient amplification and without amplification. Detailed experiments are performed on VGG-19, Resnet-18 and Resnet-34 models to analyze the impact of gradient amplification with different amplification parameters. We learn that only a




(a) VGG-19 (b) Resnet-18 (c) Resnet-34

Figure (2.14) Performance of the best models with gradient amplification over 150 epochs compared to original model with no gradient amplification. Original training(gray), testing(blue) accuracies including their mean accuracies are plotted along with amplified training(green) and testing(red) accuracies. These plots demonstrate that the models do not overfit while training with amplification.

proportion of layers are sufficient to attain such a performance gain. It can also be observed that BN layers give the best improvement while performing amplification, followed by convolution and ReLU layers, whereas the performance quickly diminishes when any combinations of these layers are used. All these experiments show that our proposed amplification method and training strategy increase the performance of the original models and achieve better accuracies even at higher learning rates.

# 3

# GRADIENT AMPLIFICATION WITH INTELLIGENT LAYER SELECTION

In this chapter, we discuss a formulated approach to determine amplification layers so that the type and the ratio of the layers to be amplified. Our proposed method along with the training strategies used for various deep learning models for CIFAR-10 and CIFAR-100 datasets are also discussed. Our main goal is to improve the performance of the models the higher values of learning rates.

## 3.1 Evolution of gradient amplification strategies

Our previous amplification method of random selection of layers has overhead of identifying the ratio of layers, the type of layers and the combination of those types of layers. With the increase in the type of layers and the number of such layers as the network becomes deeper, it is challenging to determine the best models of all the combinations. In this section, we aim to formulate a way to automatically determine the layers which need to be amplified.

### 3.1.1 Effect of gradient amplification on learning rate

The general weight update formula during training of neural networks can be written as

$$\boldsymbol{W}_{t+1} = \boldsymbol{W}_t + \eta \nabla J(\boldsymbol{W}_{t+1})$$

where $\boldsymbol{W}_t$ represents the weights of a network in the current iteration $t$, $\eta$ is the learning rate and $\nabla J(\boldsymbol{W}_{t+1})$ corresponds to the gradients of the weights computed as the derivative of error of cost function with respect to weights.

After performing gradient amplification, $\nabla J(\boldsymbol{W}_{t+1})$ gets modified depending on the

amplification factor. Let us denote the amplified gradient as

$$\nabla J_{amp} = amp * \nabla J(\boldsymbol{W}_{t+1})$$

Therefore the weight update formula after gradient amplification can be written as:

$$\begin{aligned}
\boldsymbol{W}_{t+1} &= \boldsymbol{W}_t + \eta \nabla J_{amp}(\boldsymbol{W}_{t+1}) \\
\boldsymbol{W}_{t+1} &= \boldsymbol{W}_t + \eta * amp * \nabla J(\boldsymbol{W}_{t+1}) \\
\boldsymbol{W}_{t+1} &= \boldsymbol{W}_t + (\eta * amp) * \nabla J(\boldsymbol{W}_{t+1}) \\
\boldsymbol{W}_{t+1} &= \boldsymbol{W}_t + \eta_{amp} * \nabla J(\boldsymbol{W}_{t+1}),\ where\ \eta_{amp} = \eta * amp
\end{aligned} \tag{3.1}$$

From the above analysis, one can conclude that amplifying gradients is equivalent to increasing learning rates. To determine the layers that need to be amplified, one approach is to identify the layers that learn actively during the training process. Authors of [67] propose a way to speedup the training process of deep learning models using layer freezing approach by determining the fluctuations of gradients in layers. Similar approach can be employed to determine the layers that are actively learning. As performing amplification is equivalent to increasing step size of weight updates, learning rate (or step size) can be increased when the current weights of the neurons are relatively far from optima and their gradients are all moving in the same direction to converge to optima. In addition to determining the layers that are actively learning, it is also important that the gradients of the neurons in the layers are all moving in the same direction for the amplification to be meaningful. This can be formulated by analyzing the gradients of the neurons in a layer across iterations. One simple way to perform such an identification is to compute the sum of the gradient values over iterations for all neurons in a layer.

### 3.1.2 Layer gradient directionality ratio measure, $G$

The effective direction of gradient change of the neurons in a layer $l$ can be determined using the ratio of the sum of the gradients across different iterations to its absolute gradient

sum in an epoch. Here, $m$ and $n$ correspond to the number of iterations in an epoch and the number of neurons(or weights) in a layer respectively.

$$layer\_gradient\_directionality\_ratio(G_l) = \begin{cases} 0, \ if \ \sum_{i}^{n}\sum_{j}^{m} |g_{lij}| = 0 \\ \frac{\sum_{i}^{n} |\sum_{j}^{m} g_{lij}|}{\sum_{i}^{n}\sum_{j}^{m} |g_{lij}|} \ otherwise \end{cases} \tag{3.2}$$

The above formula determines how the weights in a layer are modified. When all the weight updates of the neurons occur in the same direction across all iterations, its value is 1. When either the model reaches optimal solution (where the gradient for every neuron becomes zero) or half of the neuron weights move in the opposite direction to the other half with same magnitude (ideal case) then the value becomes 0. Otherwise, it lies in between 0 and 1. Values close to 1 signify that most of the weights are changing in the same direction and vice versa.

$$0 \leqslant G_l \leqslant 1 \tag{3.3}$$

### 3.1.3 Normalized layer gradient directionality ratio measure, $\hat{G}$

After computing the $layer\_gradient\_directionality\_ratio(G_l)$ for each layer in an epoch, it is observed that most of these values are in same range. To identify the most significant layers close to 1, $layer\_gradient\_directionality\_ratio(G_l)$ of all the layers are converted to a normal distribution using :

$$\hat{G} = \frac{G - \overline{\mathrm{G}}}{\sigma_G} \tag{3.4}$$

These normalized values signify how far they lie from the mean value. If the normalized value of a layer is close to 0, then it lies close to the mean value and its value signifies the magnitude of standard deviation(s) it is away from the mean value. Greater the positive(or negative) value farther it is from the mean. Clearly when the normalized value is a large positive value (signifying farther from the mean on the right side), it can be considered closer

to the 1 (in (3.3)), where as for large negative values, it is considered close to 0(in (3.3)).

### 3.1.4 Improved Layer gradient directionality ratio measure, $G'$

The gradients of neurons in a layer lie in a similar range and the measure $G$ has the sum of all the gradients of neurons across all the iterations, it gives higher importance to the gradients ranges that are frequent during the training. If equal importance is given to the all the neurons, then even if some of the ratio measure is close to 1, then the layer has relatively higher ratio value. With this modification, even if some of the neuron gradients in a layer are moving in the same direction, the layer has higher chance of being identified.

The effective direction of gradient change of neurons in a layer $l$ is therefore determined using the ratio of the sum of the gradients across different iterations to its absolute gradient sum for each neuron and taking the mean of these ratio values. Here, $m$ and $n$ correspond to the number of iterations in the epoch and number of neurons(or weights) in a layer respectively for an epoch.

$$improved\_layer\_gradient\_directionality\_ratio(G'_l) = \begin{cases} 0, \ if \ \sum_{j}^{m} |g_{lij}| = 0 \\ \frac{1}{n} \sum_{i}^{n} \frac{|\sum_{j}^{m} g_{ij}|}{\sum_{j}^{m} |g_{ij}|} \ otherwise \end{cases} \tag{3.5}$$

The above formula also determines how the weights in a layer are modified with equal importance given to all the neurons in a layer. When all the weight updates of the neurons occur in the same direction across all iterations, its value is 1. With the modified formula, when some of the neurons are close to 1, it have higher chances of being amplified.

$$0 \leqslant G'_l \leqslant 1 \tag{3.6}$$

### 3.1.5 Normalized layer gradient directionality ratio measure, $\hat{G'}$

After computing the $layer\_gradient\_directionality\_ratio(G'_l)$ for each layer in an epoch, in order to identify the most significant layers close to 1, $layer\_gradient\_directionality\_ratio(G'_l)$ of all the layers are converted to a normal distribution using :

$$\hat{G'} = \frac{G' - \overline{\mathrm{G'}}}{\sigma_{G'}} \tag{3.7}$$

These normalized values signify how far they lie from the mean value. If the normalized value of a layer is close to 0, then it lies close to the mean value and its value signifies the magnitude of standard deviation(s) it is away from the mean value. Greater the positive(or negative) value farther it is from the mean. Clearly when the normalized value is a large positive value (signifying farther from the mean on the right side), it can be considered closer to the 1 (in (3.6)), where as for large negative values, it is considered close to 0(in (3.6)).

### 3.1.6 Determining amplification layers using $G$,$G'$ measures

We consider the following cases to perform amplification based on thresholds. Here the formula are shown for measure $G$ and the same cases can be applied for $G'$ (by replacing $G$ with $G'$).

**Case-1: Amplify only one side ($G_l$ close to 1)** When $(G_l)$ values are close to 1, most of the weights in the layer are modified in the same direction. This suggests that the neurons are all moving down(or up) the slope to approach optima, learning rate can be increased. We amplify the gradients of the layer when its normalized value exceeds a threshold value,

$$if(\hat{G} > threshold) \ : \ amplify\ layer$$

**Case-2: Amplify both sides ($G_l$ close to 0 or 1)** As mentioned earlier, $layer\_gradient\_directionality\_ratio(G_l)$ values close to 0 either mean they are close to optima

or a plateau surface. If the weights are close to optima, adding small noise to the gradients will cause the weights to converge eventually. Otherwise, it would make the weights to cross the plateau surface and thereby improving the training process. With this assumption, we propose to amplify the gradients of a layer when the $layer\_gradient\_directionality\_ratio(G_l)$ values are close to either end (i.e., 0 or 1), we amplify whenever the absolute normalized value crosses threshold value.

$$if(|\hat{G}| > threshold) \ : \ amplify\ layer$$

Here is the overview of the function to determine amplification layers

**Algorithm 3** Determination of *amp* layers using Formula-1

**Input:** $M$, $threshold$, $g_{lij}$, $case$
**Function:** GetGradientAmpLayers($M$, $threshold$, $g_{lij}$, $case$)
  Compute $layer\_gradient\_directionality\_ratio(G_l)$ as in equation (3.2)
  Compute normalized values $\hat{G}$ as in equation (3.4)
  **if** $case == 1$ **then**
    **if** $\hat{G}_l > threshold$ **then**
      add layer $l$ to $amp$
    **end if**
  **else**
    **if** $case == 2$ **then**
      **if** $mod(\hat{G}_l) > threshold$ **then**
        add layer $l$ to $amp$
      **end if**
    **end if**
  **end if**
  **return** $amp$
**EndFunction**

### 3.2 Experiments

Experiments are performed on CIFAR-10, CIFAR-100 datasets with the similar setup to the experiments performed in Chapter 3 section 2.2. We primarily perform thorough experiments for VGG-19, Resnet-18 and Resnet-34 models. These models are trained for

150 epochs, where the learning rate of the first 100 epochs is 0.1 and the next 50 epochs is 0.01. Our earlier analysis in Chapter 3 section 2.2 show that models perform better when the amplification factor is in the range of [1.75-2.25]. We use the amplification factor 2 in our analysis. We perform experiments using the training strategy($params = [(50, 0.1, 0, 1), (100, 0.1, is\_amp, 2), (130, 0.01, is\_amp, 2), (150, 0.01, 0, 1)]$, where $is\_amp$ represents a non-zero value when amplification is performed or 0 otherwise. The values in each element in the $params$ list represent end epoch, learning rate, is_amplification_performed(non-zero) and gradient amplification factor respectively. For instance, $(50, 0.1, 0, 1)$ means that the model is trained with learning rate 0.1 until we reach 50 epochs, during which no layers are selected for gradient amplification and amplification factor is 1. In our training strategy, where $params_1 = [(50, 0.1, 0, 1), (100, 0.1, is\_amp, 2), (130, 0.01, is\_amp, 2), (150, 0.01, 0, 1)]$, no amplification is performed for the first 50 epochs and gradients in the $51^{st}$ epoch are analyzed to determine the layers to perform amplification until $100^{th}$ epoch. At epoch 101, learning rate is reduced to 0.01 and gradients are analyzed again to determine next set of amplification layers, where the selected layers are amplified for the next 29 epochs and the last 20 epochs are trained without amplification. Experiments are performed varying thresholds from 0.7 to 2.5 with the step size of 0.1. Based on the analysis of these results, we also run experiments on even deeper resnet-50 and resnet-101 models.

For deeper models, another training strategy is employed, where $params_2 = [(10, 0.1, 0, 1), (100, 0.1, is\_amp, 2), (145, 0.01, is\_amp, 2), (150, 0.01, 0, 1)]$, no amplification is performed for the first 10 epochs and the gradients in the $11^{th}$ epoch are analyzed to identify the layers to perform amplification until $100^{th}$ epoch. At epoch 101, learning rate is reduced to 0.01 and the gradients are analyzed again to determine next set of amplification layers, where the selected layers are amplified for the next 44 epochs and the last 5 epochs are trained without amplification. Experiments are performed varying thresholds from 1 to 3 with the step size of 0.25.

We also perform analysis on how frequent the amplification layers need to be varied while training the model. In our training strategy, amplification is applied from 51-100

($\eta = 0.1$) and 101-130 ($\eta = 0.01$) epochs. At first, amplification layers are determined on the onset of amplification epochs for each different learning rate and then analysis is done when the amplification layers are changed every 2 epochs from 51-130 and then every 5 epochs.

## 3.3 Results

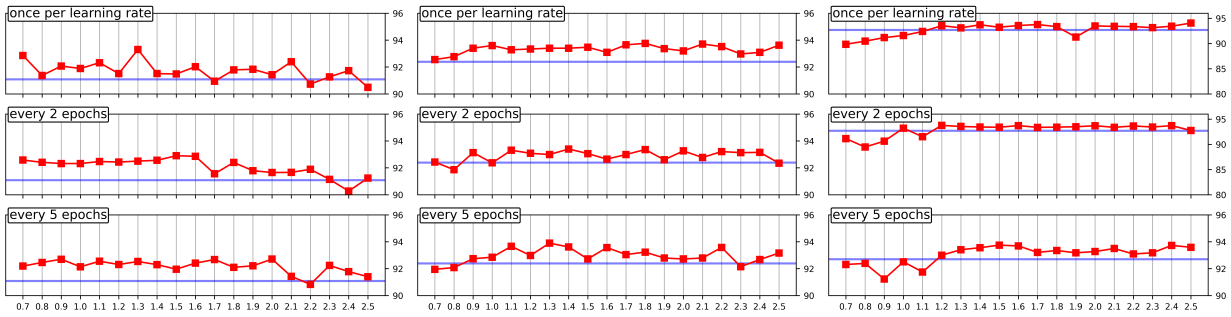
In our experiments, firstly we analyze simpler models namely, resnet-18, resnet-34 and VGG-19 models using CIFAR-10 dataset and then extend to deeper resnet architectures and also for CIFAR-100 dataset.

While training these models, a first few epochs are trained normally without any amplification. Then the gradients of the models are analyzed for an epoch by computing normalized gradient rates for all layers. As mentioned previously, for each layer it determines the rate of fluctuations of weight updates, with 1 corresponding to less fluctuations and 0 for more fluctuations. Since thresholds are measured on normalized gradient rates of layers, values beyond thresholds signify the percentage of layers being amplified and indirectly controls the ratio of amplified layers. Lower the threshold value means larger the ratio of amplified layers and vice versa. In our experiments, thresholds are varied from 0.7 - 2.5 in steps of 0.1 for simpler models and from 1.0 - 3.0 in steps of 0.25 for deeper models.

### 3.3.1 How frequently should amplification layers be modified/reseleted?

Analysis is also done on how frequent should these amplification layers be selected by running experiments when amplification layers are changed once per each learning rate, every 2 epochs and 5 epochs. Fig. 3.1 and 3.2 show the performance of vgg-19, resnet-18 and resnet-34 models when these models are amplified using case-1 and case-2 selection strategies. It can be observed that the performance improvement does not vary much for a model and one can always fine-tune to determine the best possible layer selection frequency. However, for our further analysis on CIFAR-10 dataset on deeper resnet-50, resnet-101 models, learning rate is changed once per learning rate for simplicity.

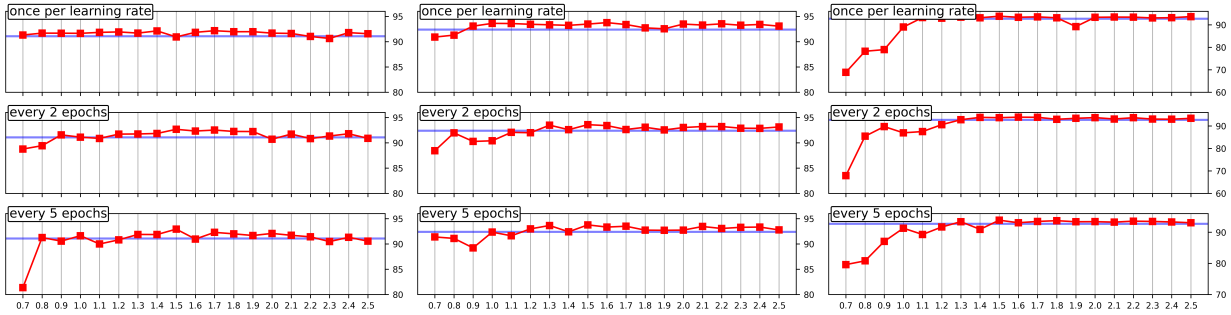
CIFAR-10 dataset, measure $G$ case-1




(a) $VGG-19$ (b) $Resnet-18$ (c) $Resnet-34$

Figure (3.1) Performance of the amplified models where layers are selected at different rates compared to mean accuracies of the original models(blue) with no gradient amplification. In each figure, we show the performance of models when amplification layers are selected once per learning rate(top), selected every 2 epochs (middle) and selected every 5 epochs(bottom). Horizontal axis refers to the thresholds applied on the normalized gradient rate ($\hat{G}_l$) using case-1 strategy and vertical axis corresponds to testing accuracies of the models.

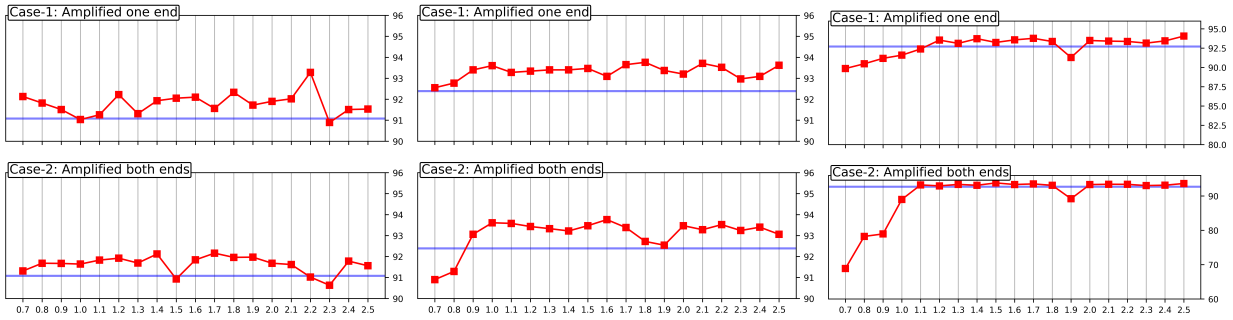
CIFAR-10 dataset, measure $G$ case-2




(a) $VGG-19$ (b) $Resnet-18$ (c) $Resnet-34$

Figure (3.2) Performance of the amplified models where layers are selected at different rates compared to mean accuracies of the original models(blue) with no gradient amplification. In each figure, we show the performance of models when amplification layers are selected once per learning rate(top), selected every 2 epochs (middle) and selected every 5 epochs(bottom). Horizontal axis refers to the thresholds applied on the normalized gradient rate ($\hat{G}_l$) using case-2 strategy and vertical axis corresponds to testing accuracies of the models.

CIFAR-10 dataset, measure $G$



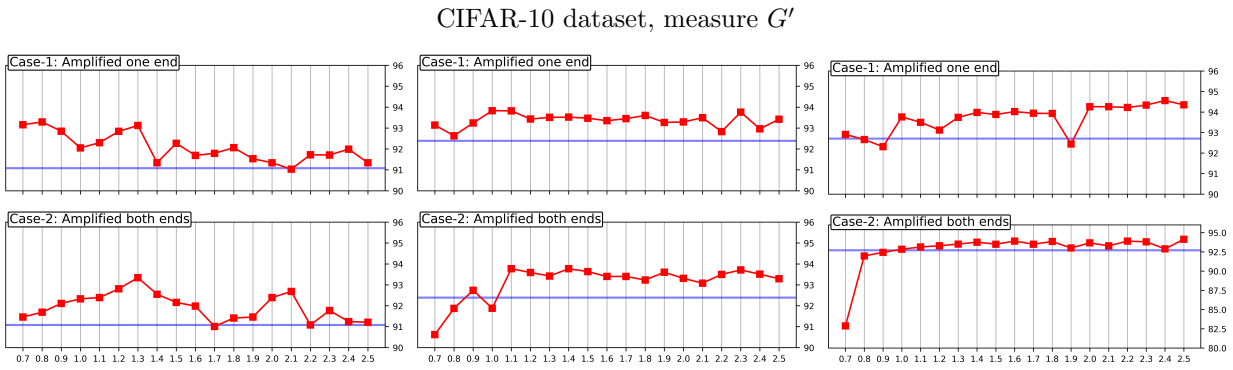

(a) $VGG-19$ (b) $Resnet-18$ (c) $Resnet-34$

Figure (3.3) Performance of the models on CIFAR-10 dataset with amplified models using $G$ applied from epochs 51-130 compared to mean accuracies of the original models(blue) with no gradient amplification. Horizontal axis refers to the thresholds applied on the normalized gradient rate ($\hat{G}_l$) and vertical axis corresponds to testing accuracies of the models.

### 3.3.2 Analysis on CIFAR-10 dataset

**Analysis on simpler models** Fig. 3.3, 3.4 show the performance of VGG-19, resnet-18 and resnet-34 models for a range of thresholds 0.7-2.5 with a step-size of 0.1 when $G$ and $G'$ amplification is applied respectively. For VGG-19 models, when amplified using case-1 and case-2 strategies, accuracies improve for most of the threshold values. Though there is no defined pattern in either case while using $G$, models seem to perform better with lower threshold values less than 1.5 in case of $G'$. Resnet-18 model perform better while using $G$ or $G'$ formula. In case-1 strategy, models perform for all the threshold values, but for case-2 strategy, models with thresholds greater than 1 perform better than original models. In the case of resnet-34 models, models perform better for thresholds greater than 1.2 and case-1 strategy has better improvements over case-2 in both $G$ and $G'$. Also, accuracy of the resnet-34 models drop for threshold value of 1.9 in all the cases, otherwise accuracies of the models increase slightly with the increasing thresholds.

**Analysis on deeper models** As deeper models have longer training times, amplification is performed on reduced thresholds ranging from 1.0 - 3.0 in steps of 0.25. Fig. 3.5 and 3.6 show the testing accuracies of the resnet-50 and resnet-101 models for a




(a) VGG-19 (b) Resnet-18 (c) Resnet-34

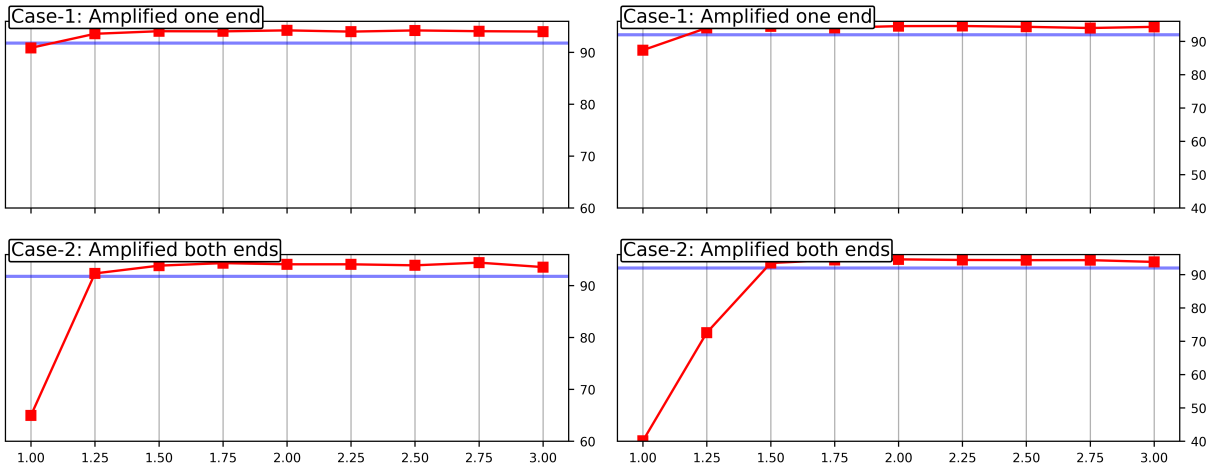
Figure (3.4) Testing accuracies(Y-axis) of the amplified models(red) using $G'$ compared to mean accuracies of the original models(blue) with no gradient amplification for a range of thresholds(X-axis) applied on the normalized gradient rate ($\hat{G}'_l$) on CIFAR-10 dataset.

range of thresholds respectively. For both $G$ and $G'$ while using case-1 strategy, amplified models perform close to original models for lower thresholds and as the threshold value increases amplified models always perform better than original models. Models with case-1 strategy seems to be robust compared to case-2 for lower thresholds. While using case-2 strategy, for lower thresholds, amplified models perform lower than the original models but the performance of the amplified models increase with increasing thresholds. This suggests that deeper models perform better with higher threshold values. Models with case-1 as amplification strategy perform better for thresholds more than 1.25, where as for case-2, models with thresholds 1.5 perform better.

### 3.3.3 Analysis on CIFAR-100 dataset

To emphasize the generality of amplification, experiments are also performed on CIFAR-100 dataset. In these experiments, amplification layers are selected once per each learning rate in the training epochs while using previously mentioned training strategy $params = [(50, 0.1, 0, 1), (100, 0.1, is\_amp, 2), (145, 0.01, is\_amp, 2), (150, 0.01, 0, 1)]$. Experiments are performed with both gradient change measures $G$ and $G'$ using case-1 and case-2 layer selection strategies.

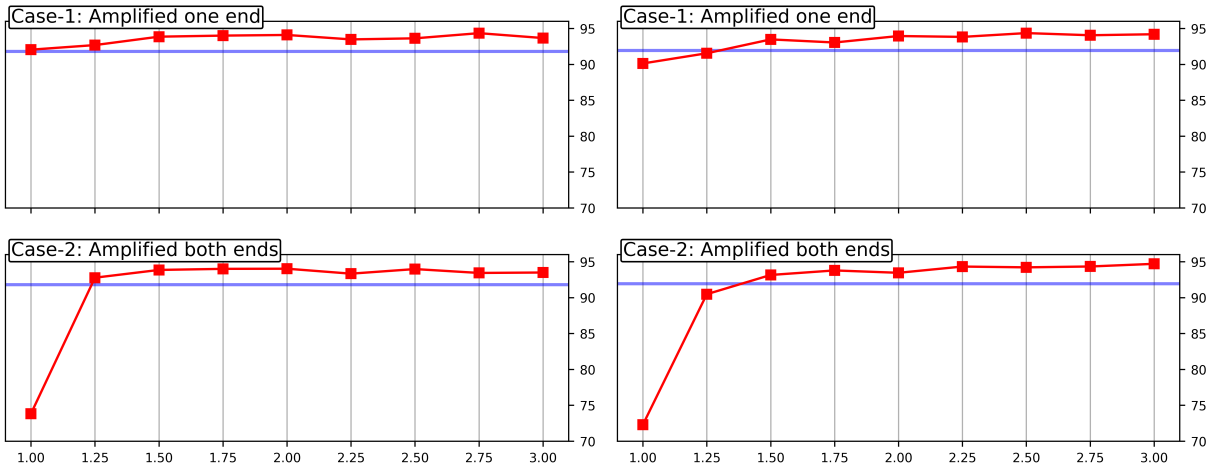
CIFAR-10 dataset, measure $G$




(a) Resnet-50 (b) Resnet-101

Figure (3.5) Testing accuracies(Y-axis) of resnet-50 and resnet-101 models with $G_l$ layer amplification(red) applied from epochs 51-145 compared to mean accuracies of the original models(blue) with no gradient amplification for a range of threshold values(X-axis).

CIFAR-10 dataset, measure $G'$



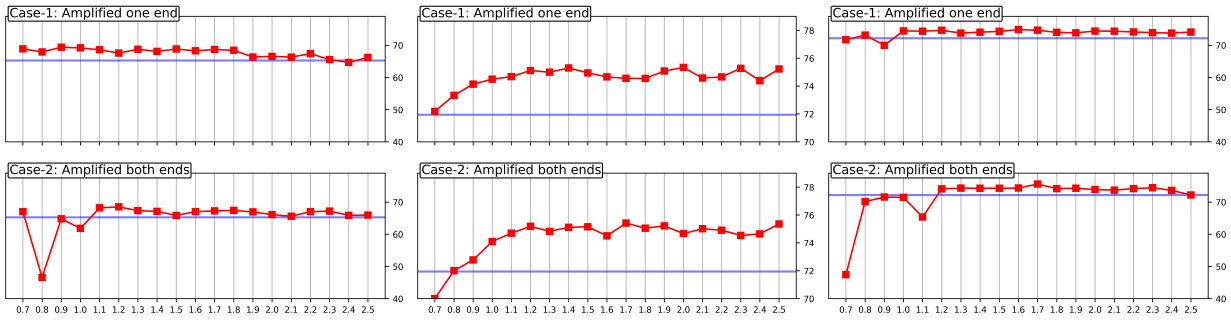

(a) Resnet-50 (b) Resnet-101

Figure (3.6) (CIFAR-10 dataset) Testing accuracies(Y-axis) of resnet-50 and resnet-101 models with $G'_l$ layer amplification(red), applied from epochs 51-145, compared to mean accuracies of the original models(blue) with no gradient amplification for a range of threshold values(X-axis).

**Analysis on simpler models** Fig. 3.7, 3.8 show the performance of VGG-19, resnet-18 and resnet-34 models for a range of thresholds 0.7-2.5 with a step-size of 0.1 when $G$ and $G'$ amplification is applied respectively. For VGG-19 models, when case-1 amplification method is used, for lower thresholds with both $G$ and $G'$, models perform better at lower threshold values but have similar performance to original models in the case of higher thresholds. When case-2 is used,performance of the models is sensitive to threshold values. Models have lower performance than original models for small thresholds and have similar performance for large thresholds. For intermediate threshold values, it either has better or similar performance to original models. For resnet-18, when case-1 is used, for all the models have better performance than the original models for all the thresholds. Performance of the models increase with the thresholds around 1.5 and then the performance improvement remains the same. While for case-2, models have reduced performance for lower thresholds upto 0.9 in $G$ and 1.1 in $G'$ which then increases until 1.5 and then the improvement remains the same. Resnet-34 models also have similar performance behavior as resnet-18 models maintaining the improved performance with increasing thresholds for case-2. While for case-1,for both $G$ and $G'$ models until threshold reaches 1.2, accuracies of the models increase but are lower than the original models and improve than original after 1.2 maintaining the improved accuracy with the increasing thresholds.

**Analysis on deeper models** For deeper networks, resnet-50 and resnet-101, amplification is performed on reduced thresholds ranging from 1.0 - 3.0 in steps of 0.25. Fig. 3.9 and 3.9 show the testing accuracies of the resnet-50 and resnet-101 models for a range of thresholds. In resnet-50 models, for both $G$ and $G'$ while using case-1 strategy, amplified models always perform better than original models and the improvement of the accuracies almost remain the same across threshold values. While for case-2 strategy, models have lower performance for threshold value 1.00 and the testing accuracies are better than the original models from threshold 1.25. In resnet-101 models, for both $G$ and $G'$ w amplified models perform better from thresholds 1.25 and 1.50 respectively while using case-1 and

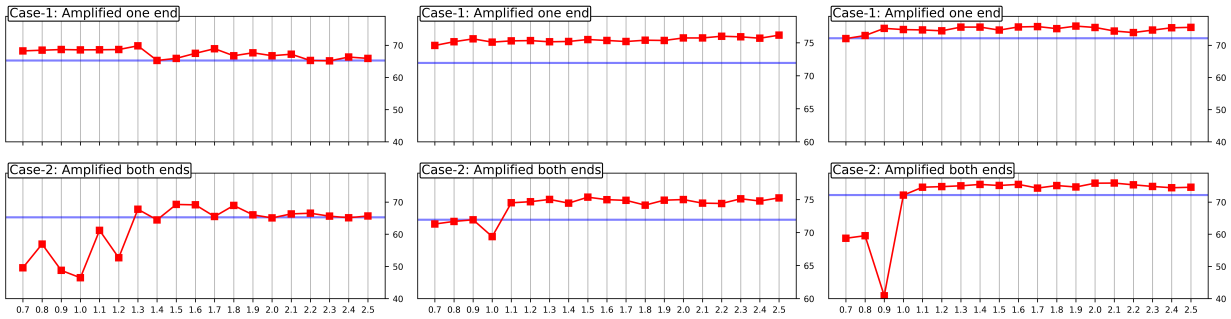
CIFAR-100 dataset, measure $G$




(a) $VGG-19$ (b) $Resnet-18$ (c) $Resnet-34$

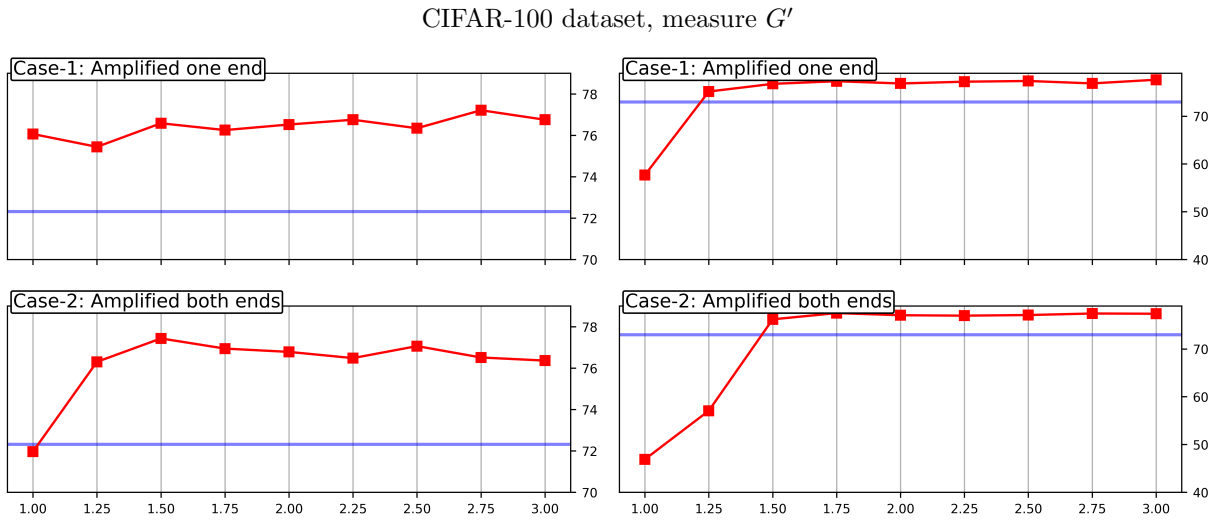
Figure (3.7) Performance of the models on CIFAR-100 dataset with amplified models(red) using $G$ applied from epochs 51-130 compared to mean accuracies of the original models(blue) with no gradient amplification. Horizontal axis refers to the thresholds applied on the normalized gradient rate ($\hat{G}_l$) and vertical axis corresponds to testing accuracies of the models.

CIFAR-100 dataset, measure $G'$




(a) $VGG-19$ (b) $Resnet-18$ (c) $Resnet-34$

Figure (3.8) Testing accuracies(Y-axis) of the amplified models(red) using $G'$ compared to mean accuracies of the original models(blue) with no gradient amplification for a range of thresholds(X-axis) applied on the normalized gradient rate ($\hat{G}'_l$) on CIFAR-100 dataset.




(a) Resnet-50 (b) Resnet-101

Figure (3.9) (CIFAR-100 dataset) Testing accuracies(Y-axis) of resnet-50 and resnet-101 models with $G_l$ layer amplification(red) applied from epochs 51-145 compared to mean accuracies of the original models(blue) with no gradient amplification for a range of threshold values(X-axis).

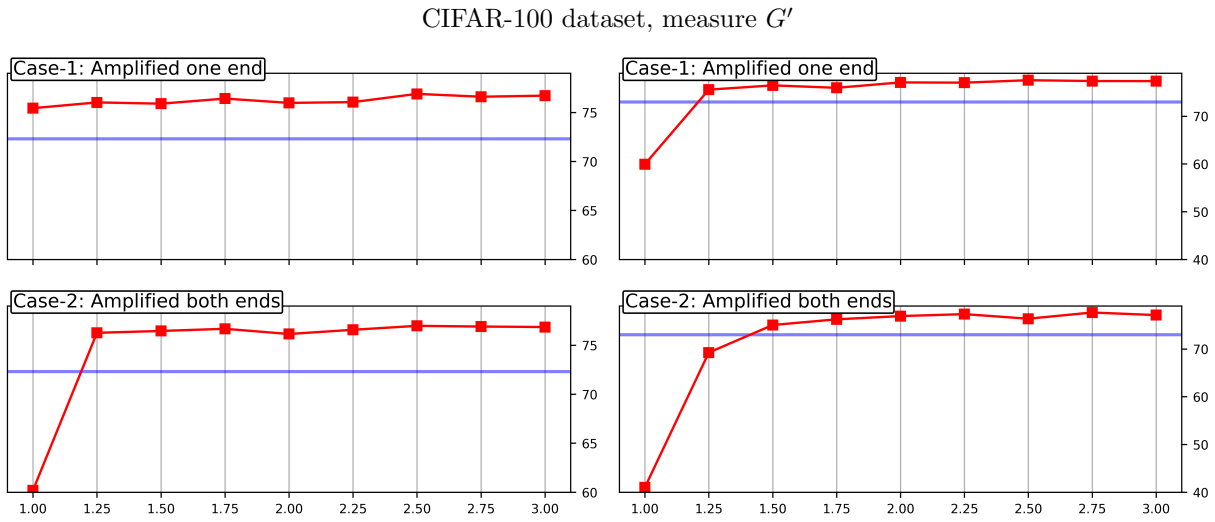
case-2 strategy. Though the improvement in the performance appears the same, testing accuracies improve slowly with increasing thresholds.

### 3.3.4 Comparison of running times

Table 3.1 and 3.2 show the mean running times(in minutes) across 10 runs of original models and amplified models for different ratio measures and cases. Training is performed on the GSU high performance cluster with NVIDIA V100 GPUs with only our models running on the GPUs with no other user jobs. Performing amplification while training increases the training times by only 1-3 minutes for all the models in most of the cases. Therefore, training models performing amplification improves the accuracy of the models maintaining training times closer (less than 2% increment) to original models.

### 3.3.5 Best models

Here, we compare the best results of amplified models in each case with their corresponding original models without amplification. Testing accuracies of the best amplified




(a) Resnet-50 (b) Resnet-101

Figure (3.10) (CIFAR-10 dataset) Testing accuracies(Y-axis) of resnet-50 and resnet-101 models with $G'_l$ layer amplification(red) applied from epochs 51-145 compared to mean accuracies of the original models(blue) with no gradient amplification for a range of threshold values(X-axis).

Table (3.1) Mean running times(in minutes) of models with $G, G'$ layer-based gradient amplification on CIFAR-10 dataset across 10 iterations.

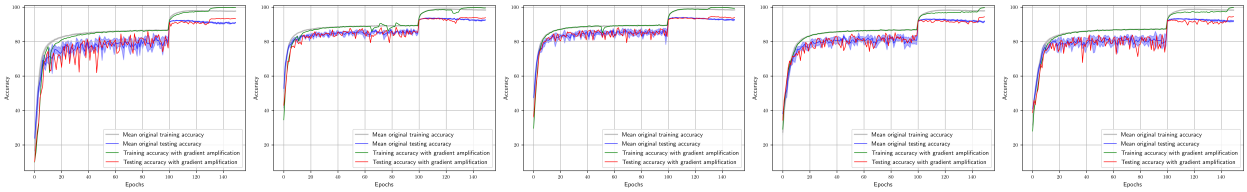
| Model | *Original* (*min*) | *Ours* using $\hat{G}(min)$ | | *Ours* using $\hat{G'}(min)$ | |
|---|---|---|---|---|---|
| | | Case-1 | Case-2 | Case-1 | Case-2 |
| *VGG_19* | 31.35 ± 0.45 | 31.82 ± 0.52 (**1.5%**) | 31.94 ± 0.45 (**1.89%**) | 31.87 ± 0.45 (**1.66%**) | 31.82 ± 0.35 (**1.5%**) |
| *Resnet_18* | 42.15 ± 0.23 | 42.79 ± 0.45 (**1.53%**) | 42.73 ± 0.46 (**1.38%**) | 42.96 ± 0.57 (**1.92%**) | 42.76 ± 0.44 (**1.45%**) |
| *Resnet_34* | 66.35 ± 0.92 | 67.73 ± 0.95 (**2.09%**) | 67.64 ± 0.9 (**1.94%**) | 67.8 ± 0.82 (**2.18%**) | 67.68 ± 0.91 (**2.01%**) |
| *Resnet_50* | 139.64 ± 1.97 | 139.71 ± 2.01 (**0.05%**) | 140.07 ± 1.59 (**0.31%**) | 140.63 ± 1.29 (**0.71%**) | 141.06 ± 1.05 (**1.01%**) |
| *Resnet_101* | 224.22 ± 3.45 | 225.48 ± 2.89 (**0.56%**) | 226.37 ± 2.07 (**0.96%**) | 227.26 ± 2.12 (**1.35%**) | 227.78 ± 2.02 (**1.58%**) |

models are shown in the table 3.3 and 3.4 for CIFAR-10 and CIFAR-100 datasets. Training and testing accuracies for each epoch of these best models with amplification along with the original models are shown in Fig. 3.11, 3.12 for CIFAR-10 dataset and in Fig. 3.13, 3.13 for CIFAR-100 dataset. Since the mean accuracy of the original models are compared, training accuracies(in gray), testing accuracies (in blue) including their mean accuracies are plotted along with amplified training(in green) and testing(in red) accuracies. These plots signify the importance of having the final epochs of the model to be trained without amplification and also demonstrate that the models do not overfit while training with amplification.

Table (3.2) Mean running times(in minutes) of models with $G, G'$ layer-based gradient amplification on CIFAR-100 dataset across 10 iterations.

| Model | *Original* (*min*) | *Ours* using $\hat{G}$ (*min*) | | *Ours* using $\hat{G}'$ (*min*) | |
|---|---|---|---|---|---|
| | | Case-1 | Case-2 | Case-1 | Case-2 |
| *VGG_19* | 31.62 ± 0.48 | 32.55 ± 0.43 (**2.92%**) | 31.92 ± 0.49 (**0.92%**) | 32.16 ± 0.69 (**1.7%**) | 31.83 ± 0.57 (**0.64%**) |
| *Resnet_18* | 42.08 ± 0.47 | 42.83 ± 0.64 (**1.78%**) | 42.24 ± 0.69 (**0.39%**) | 43.44 ± 2.65 (**3.23%**) | 42.19 ± 0.57 (**0.26%**) |
| *Resnet_34* | 66.64 ± 0.84 | 67 ± 0.89 (**0.54%**) | 66.99 ± 0.96 (**0.53%**) | 69.84 ± 6.11 (**4.8%**) | 66.88 ± 0.9 (**0.36%**) |
| *Resnet_50* | 139.32 ± 1.18 | 140.64 ± 0.51 (**0.95%**) | 140.21 ± 1.7 (**0.64%**) | 140.68 ± 0.51 (**0.97%**) | 140.79 ± 0.68 (**1.06%**) |
| *Resnet_101* | 223.88 ± 2.69 | 226.35 ± 3.05 (**1.10%**) | 226.28 ± 1.87 (**1.07%**) | 225.09 ± 3.23 (**0.54%**) | 226.12 ± 2.18 (**1.00%**) |

CIFAR-10 dataset, measure $G$



(a) VGG-19 (b) Resnet-18 (c) Resnet-34 (d) Resnet-50 (e) Resnet-101

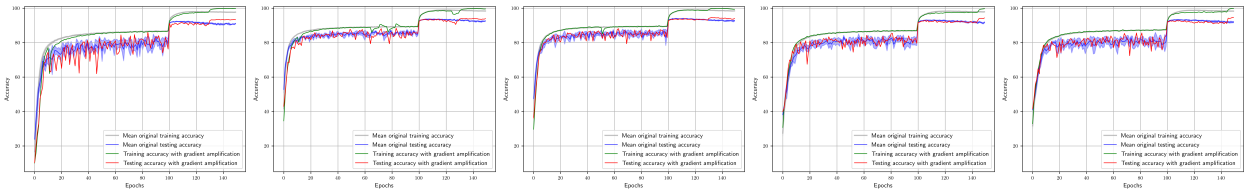
Figure (3.11) Testing accuracies(Y-axis) of the best models on CIFAR-10 dataset with amplification performed using $G$ algorithm (3) compared to original models without amplification. Original training(gray), testing(blue) accuracies including their mean accuracies are plotted along with amplified training(green) and testing(red) accuracies.

We also perform random amplification for deeper resnet models using some of the hyperparameters which have the better performance for resnet-18, resnet-34 models. The best accuracies of these models are also compared in the table below for both CIFAR-10 and CIFAR-100 datasets. Our results with amplification based on $G$ and $G'$ have similar performance and sometimes improved for VGG-19, resnet-18 and resnet-34 models on CIFAR-10 dataset. Resnet-50 and resnet-101 more than 2% improvement than original as well as randomly amplified models. In the case of CIFAR-100 dataset, all the models based on $G$ and $G'$ have significant performance improvement compared to original and random amplification.

## 3.4 Summary

We propose various measures to compute effective gradient direction of a layer during its training process. These measures are used to determine the amplification layers based

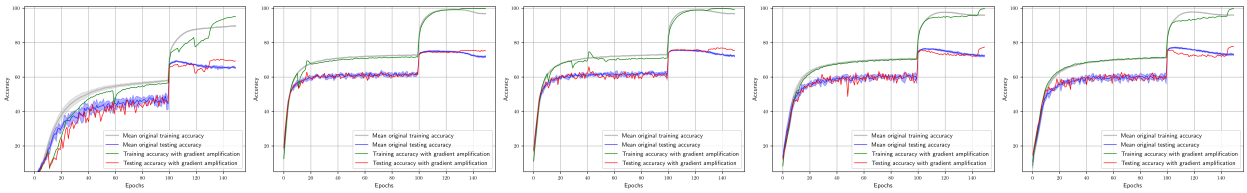
CIFAR-10 dataset, measure $G$



(a) VGG-19 (b) Resnet-18 (c) Resnet-34 (d) Resnet-50 (e) Resnet-101

Figure (3.12) Testing accuracies(Y-axis) of the best models on CIFAR-10 dataset with layer amplification performed using $G'$ compared to original models without amplification. Original training(gray), testing(blue) accuracies including their mean accuracies are plotted along with amplified training(green) and testing(red) accuracies.

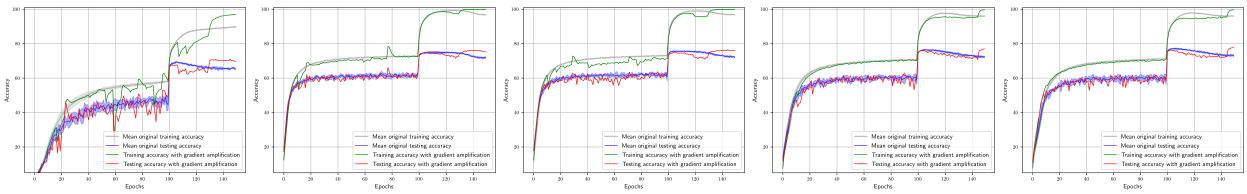
CIFAR-100 dataset, measure $G'$



(a) VGG-19 (b) Resnet-18 (c) Resnet-34 (d) Resnet-50 (e) Resnet-101

Figure (3.13) Testing accuracies(Y-axis) of the best models on CIFAR-100 dataset with amplification performed using $G$ algorithm (3) compared to original models without amplification. Original training(gray), testing(blue) accuracies including their mean accuracies are plotted along with amplified training(green) and testing(red) accuracies.

CIFAR-100 dataset, measure $G'$



(a) VGG-19 (b) Resnet-18 (c) Resnet-34 (d) Resnet-50 (e) Resnet-101

Figure (3.14) Testing accuracies(Y-axis) of the best models on CIFAR-10 dataset with layer amplification performed using $G'$ compared to original models without amplification. Original training(gray), testing(blue) accuracies including their mean accuracies are plotted along with amplified training(green) and testing(red) accuracies.

Table (3.3) Performance comparison of random, $G$, $G'$ layer-based gradient amplification models on CIFAR-10 dataset.

| Model | *Original* | *Randomamp* | *Ours* using $\hat{G}$ | | *Ours* using $\hat{G}'$ | |
|---|---|---|---|---|---|---|
| | | | Case-1 | Case-2 | Case-1 | Case-2 |
| *VGG_19* | 91.08% | 93.35 % (+**2.27%**) | 93.30% (+**2.22%**) | 92.92% (+**1.84%**) | 93.29% (+**2.21%**) | 93.34% (**2.26+%**) |
| *Resnet_18* | 92.39% | 94.57% (+**2.18%**) | 93.90% (+**1.51%**) | 93.76% (+**1.37%**) | 94.49% (+**2.1%**) | 94.1% (+**1.71%**) |
| *Resnet_34* | 92.71% | 94.39%(+**1.68%**) | 94.05% (+**1.34%**) | 93.89% (+**1.18%**) | 94.56% (+**1.85%**) | 94.14% (+**1.43%**) |
| *Resnet_50* | 91.80% | 92.68% (+**0.88%**) | 94.24% (+**2.44%**) | 94.43% (+**2.63%**) | 94.34% (+**2.54%**) | 94.02% (+**2.22%**) |
| *Resnet_101* | 91.95% | 93.04% (+**1.09%**) | 94.57% (+**2.62%**) | 94.54% (+**2.59%**) | 94.35% (+**2.4%**) | 94.7% (+**2.75%**) |

Table (3.4) Performance comparison of random, $G$, $G'$ layer-based gradient amplification models on CIFAR-100 dataset.

| Model | *Original* | *Randomamp* | *Ours* using $\hat{G}$ | | *Ours* using $\hat{G}'$ | |
|---|---|---|---|---|---|---|
| | | | Case-1 | Case-2 | Case-1 | Case-2 |
| *VGG_19* | 65.27% | 66.52% (+**1.25%**) | 69.38% (+**4.11%**) | 68.5% (+**3.23%**) | 69.83% (+**4.56%**) | 69.25% (**3.98+%**) |
| *Resnet_18* | 71.94% | 72.7% (+**0.760%**) | 75.33% (+**3.39%**) | 75.41% (+**3.47%**) | 76.14% (+**4.2%**) | 75.35% (+**3.41%**) |
| *Resnet_34* | 72.18% | 73.02% (+**0.84%**) | 74.86% (+**2.68%**) | 75.59% (+**3.41%**) | 75.95% (+**3.77%**) | 75.9% (+**3.72%**) |
| *Resnet_50* | 72.32% | 73.05% (+**0.73%**) | 77.21% (+**4.89%**) | 77.43% (+**5.11%**) | 76.89% (+**4.57%**) | 76.97% (+**4.65%**) |
| *Resnet_101* | 73.00% | 73.72% (+**0.72%**) | 77.63% (+**4.63%**) | 77.51% (+**4.51%**) | 77.53% (+**4.53%**) | 77.63% (+**4.63%**) |

on two amplification thresholding strategies. Detailed experiments are performed to analyze each of the measures, their amplification strategies for a range of thresholds. Experiments are run on CIFAR-10 and CIFAR-100 datasets.

# 4

# INTELLIGENT GRADIENT AMPLIFICATION FOR NEURONS

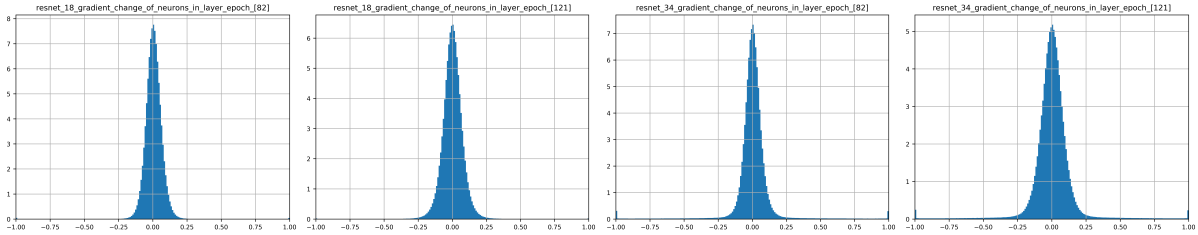
In this chapter, using the layer-based gradient measure $G'$ discussed in chapter 3, a neuron based gradient measure is formulated. This measure is used to identify the neurons in a layer that need to be amplified instead of the amplifying all the neurons in a layer. Our proposed measure is evaluated on various deep learning models for CIFAR-10 using the training discussed in the previous layer based amplification methods. We aim to determine if neuron based amplification is sufficient to achieve performance improvements at higher learning rates.

### 4.0.1 Neuron gradient directionality ratio measure, $G_n$

Based on the ratio in the definition of $G_n$ measure, the effective direction of gradient change for a neuron $n$ of a layer $l$ in a training epoch is given as

$$neuron\_gradient\_directionality\_ratio(G_n^{(l)}) = \begin{cases} 0, \ if \ \sum_i^n \sum_j^m |g_{nj}^{(l)}| = 0 \\ \frac{|\sum_j^m g_{nj}^{(l)}|}{\sum_j^m |g_{nj}^{(l)}|}, \ otherwise \end{cases} \tag{4.1}$$

where $m$ represents the number of iterations in an epoch. Here, we essentially trace the gradient updates of a neuron across all the iterations in a training epoch to determine its effective gradient update direction. If the updates on the neuron weights across all the iterations occur in the same direction, it implies the neuron is actively trying to approach the optima and therefore the learning rate can be increased ( equivalent to amplifying gradients as shown in eq. 3.1) to speed up the training. When all the weight updates of a neuron occur in the same direction across all iterations, its value is 1. When either the model reaches optimal solution (where the gradient for a neuron becomes zero) or half of the neuron weight




(a) Resnet-18 (epoch 82) (b) Resnet-18 (epoch 121) (c) Resnet-34 (epoch 82) (d) Resnet-34 (epoch 121)

Figure (4.1) Distribution of neuron gradient directionality ratio $G_n$ of all the neurons at epochs 82 and 121 for resnet models

updates move in the opposite direction to the other half with same magnitude (ideal case) then the value becomes 0. Otherwise, it lies in between 0 and 1. Values close to 1 signify that most of the weights are changing in the same direction and viceversa.

$$0 \leqslant G_n \leqslant 1 \tag{4.2}$$

### 4.0.2 Normalizing neuron gradient directionality ratio measure

Normalized neuron gradient directionality ratio measure is computed similar to the normalized layer based measures. The distribution of these neuron gradient ratio values for all the neurons across all the layers as shown in Fig. 4.1 for resnet-18 and resnet-34 models. Plots are shown for epochs 82 (where learning rate is 0.1) and 121(where learning rate is 0.01), but similar pattern exists for other epochs. These figures depict that the ratio values of neurons follow a normal distribution with mean close to 0 and width of the distribution varies across epochs depending on factors such as learning rate, type of dataset and so on. As the neurons in a layer are related architecturally, normalization can also be done using all the neurons in a layer independently for different layers. Since each of the neuron gradient directionality ratio measures are computed independently, normalization can be done either in a local way using only the neuron gradient ratio measures within the layer or in a global way, using ratio values of all the neurons across all the layers.

**Local normalization:** In this approach, gradient ratio values $G_n$ of neurons in a layer are considered for normalization, i.e., each layer independently determines the normalized gradient ratio values of its neurons. Here, $\overline{\mathrm{G}}_n^{(l)}$ and $\sigma_{G_n^{(l)}}$ represent the mean and standard deviation of the gradient ratio values of the neurons of layer $l$ respectively.

$$local - \hat{G}_n^{(l)} = \frac{G_n^{(l)} - \overline{\mathrm{G}}_n^{(l)}}{\sigma_{G_n^{(l)}}} \tag{4.3}$$

**Global normalization:** In this method, gradient ratio values $G_n$ of all neurons across all the layers are considered to transform to a normal distribution. $\overline{\mathrm{G}}_n$ and $\sigma_{G_n}$ represent the mean and standard deviation of the all neuron values in the network respectively.

$$global - \hat{G}_n^{(l)} = \frac{G_n^{(l)} - \overline{\mathrm{G}}_n}{\sigma_{G_n}} \tag{4.4}$$

### 4.0.3 Determining amplification neurons using $G_n$ measure

After analyzing the gradients of all the neurons in an epoch, normalized ratio values are computed and neurons to be amplified are determined using threshold values. From the layer based amplification, models perform better whenever layers with ration values close to 1 (case-1) are amplified. Therefore the neurons meeting the following criteria are selected for amplification in the case of local normalization.

$$if(local - \hat{G}_n > threshold) \ : \ amplify\ neuron$$

Similarly, when global normalization is performed, the following condition is used to identify the neurons whose gradients need to be amplified.

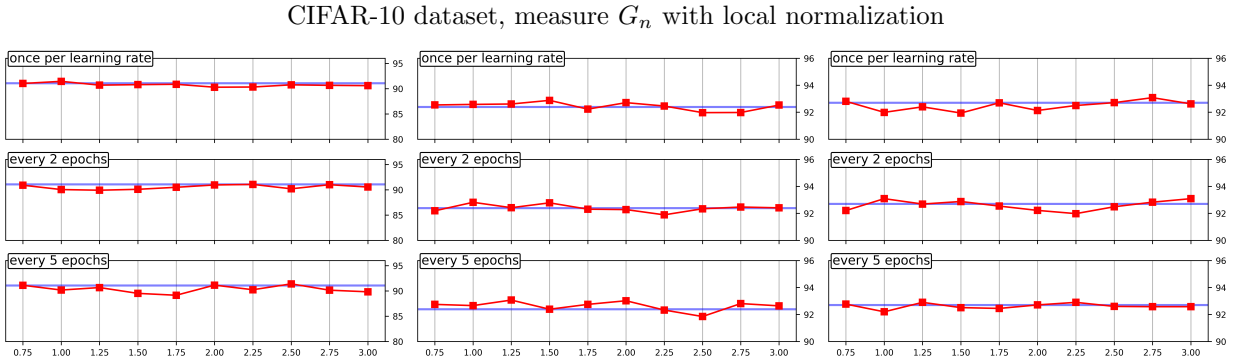
$$if(global - \hat{G}_n > threshold) \ : \ amplify\ neuron$$

### 4.1 Experiments & Results

Experiments are performed on CIFAR-10 dataset with the similar setup to the experiments performed in Chapter 2 section 2.2. We primarily perform thorough experiments for VGG-19, Resnet-18 and Resnet-34 models. These models are trained for 150 epochs, where the learning rate of the first 100 epochs is 0.1 and the next 50 epochs is 0.01. We perform experiments using two different training strategies($params$), namely $[(50, 0.1, 0, 1), (100, 0.1, is\_amp, 2), (130, 0.01, is\_amp, 2), (150, 0.01, 0, 1)]$ and $[(10, 0.1, 0, 1), (100, 0.1, is\_amp, 2), (145, 0.01, is\_amp, 2), (150, 0.01, 0, 1)]$, where $is\_amp$ represents a non-zero value when amplification is performed or 0 otherwise. Neurons to amplify are selected once per each learning rate and those neuron gradients will be amplified until the amplification factor is changed back to 1 (as shown in $params$).

In the first training strategy with $params_1 = [(50, 0.1, 0, 1), (100, 0.1, is\_amp, 2), (130, 0.01, is\_amp, 2), (150, 0.01, 0, 1)]$, where $is\_amp \neq 0$, no amplification is performed for the first 50 epochs and gradients in the $51^{st}$ epoch are analyzed to determine the neurons to perform amplification until $100^{th}$ epoch. At epoch 101, learning rate is reduced to 0.01 and gradients are analyzed again to determine next set of neurons for amplification, where the selected neurons are amplified for the next 29 epochs and the last 20 epochs are trained without amplification. Experiments are performed varying thresholds from 0.7 to 2.5 with the step size of 0.1.

In the second training strategy, where $params_2 = [(10, 0.1, 0, 1), (100, 0.1, is\_amp, 2), (145, 0.01, is\_amp, 2), (150, 0.01, 0, 1)]$, no amplification is performed for the first 10 epochs and the gradients in the $11^{th}$ epoch are analyzed to identify the neurons to perform amplification until $100^{th}$ epoch. At epoch 101, learning rate is reduced to 0.01 and the gradients are analyzed again to determine next set of neurons for amplification and these selected neuron gradients are amplified for the next 44 epochs and the last 5 epochs are trained without amplification. Thresholds are varied from 0.7 to 2.5 with the step size of 0.1 for detailed analysis.



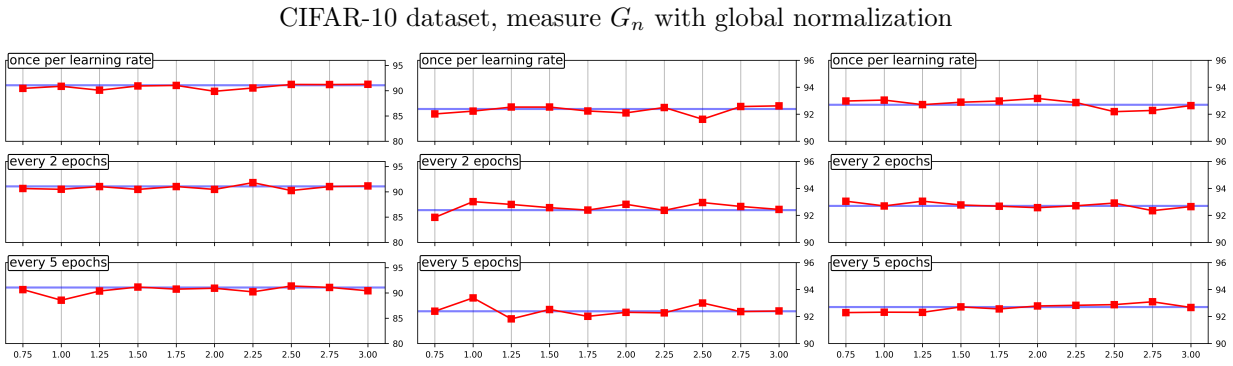

(a) $VGG-19$ (b) $Resnet-18$ (c) $Resnet-34$

Figure (4.2) Performance of the amplified models where neurons are selected at different rates compared to mean accuracies of the original models(blue) with no gradient amplification. In each figure, we show the performance of models when neurons to be amplified are selected once per learning rate(top), selected every 2 epochs (middle) and selected every 5 epochs(bottom). Horizontal axis refers to the thresholds applied on the normalized gradient rate ($\hat{G}_n$) using local normalization and vertical axis corresponds to testing accuracies of the models.

In the case of random amplification, models perform better when amplification factor lie in the range of [1.75-2.25]. As only some of the neurons in the layers are amplified, we again study the impact of amplification factor when it is increased to 2,3,4 and 5 on the *params*(training strategy) that has better performance.

### 4.1.1 How frequently should amplification neurons be modified/reseleted?

As only a few neurons are selected in a layer for amplification, analysis is done to understand how frequent should these amplification neurons be selected. We perform similar analysis as before by running experiments when amplification layers are changed once per each learning rate, every 2 epochs and 5 epochs. Fig. 4.2 and 4.3 show the performance of vgg-19, resnet-18 and resnet-34 models when these models are amplified using $\hat{G}_n$ with local and global normalization. It can be observed that the performance improvement doesnot vary much for a model and one can always finetune to determine the best possible layer selection frequency. However, for our further analysis on CIFAR-10 dataset on deeper resnet-50, resnet-101 models, learning rate is changed once per learning rate for simplicity.




(a) $VGG-19$ (b) $Resnet-18$ (c) $Resnet-34$

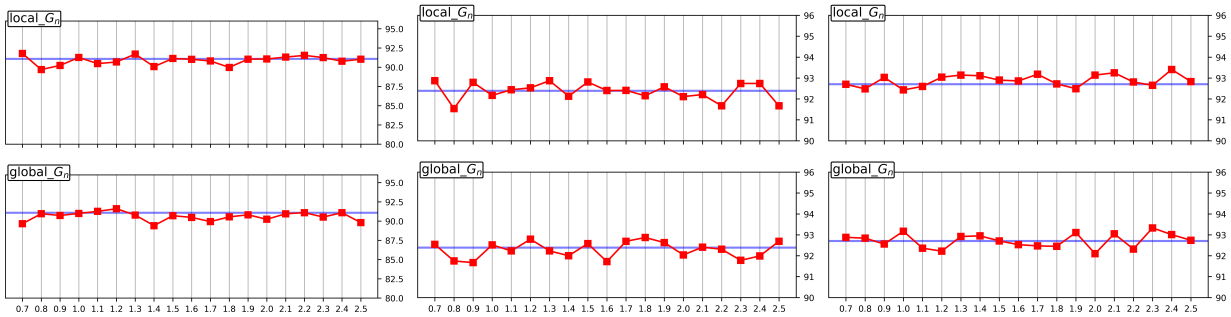
Figure (4.3) Performance of the amplified models where neurons are selected at different rates compared to mean accuracies of the original models(blue) with no gradient amplification. In each figure, we show the performance of models when neurons to be amplified are selected once per learning rate(top), selected every 2 epochs (middle) and selected every 5 epochs(bottom). Horizontal axis refers to the thresholds applied on the normalized gradient rate ($\hat{G}_n$) using local normalization and vertical axis corresponds to testing accuracies of the models.

### 4.1.2 Analysis on CIFAR-10 dataset

Fig. 4.4 and Fig. 4.5 show the performance of the models when neuron level amplification is applied for the first (during epochs 51-130) and second (during epochs 51-145) training strategies respectively. Though models have better performance for are some thresholds than original(no amplification) models, most of the threshold values have neuron amplified models having similar performance and sometimes even with reduced performance than original models. Selection of training strategies also do not have much significance on the model performance. So far, all the analysis is done on experiments with amplification factor as 2. We now try to experiment with increased amplification factor to determine if this improves the performance.

**Analysis on increased amplification factor** While performing random amplification, it is observed that models perform better when amplification factor lie in the range of [1.75-2.25]. As some of the neurons in a layer are amplified, experiments are run with increased amplification factors from 3,4,5 and the performance of these models are

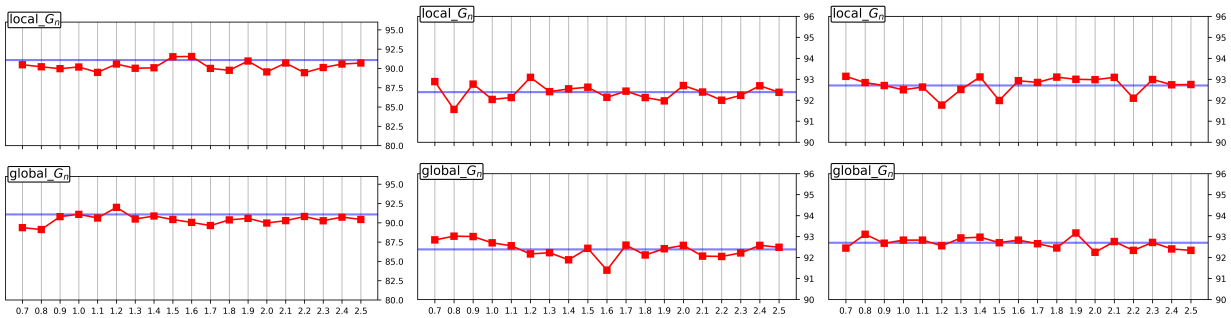
CIFAR-10 dataset, measure $G_n$




(a) VGG-19 (b) Resnet-18 (c) Resnet-34

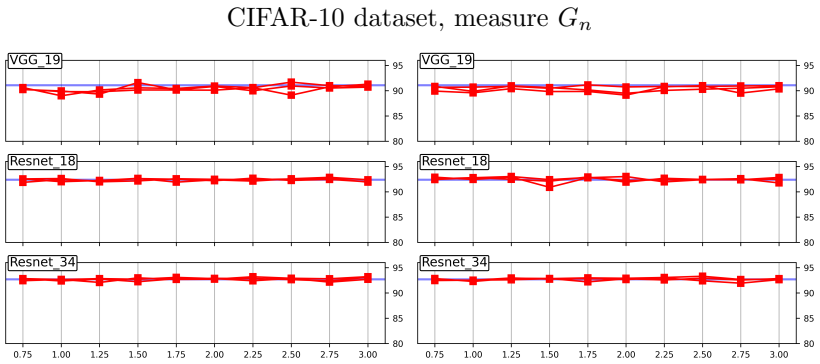
Figure (4.4) Performance of the neuron amplified models(red) using $params_1$ where epochs 51-130 are used for amplification using formula 4.1 compared to mean accuracies of the original models(blue) with no gradient amplification. For each model (a), (b), (c), the upper plot corresponds to the testing accuracies of the models when amplification is done with local-normalization approach and the lower plot corresponds when the layers to global-normalization. Horizontal axis refers to the thresholds applied on the normalized gradient rate ($\hat{G}_n$) and veritical axis correspond to testing accuracies of the model.

CIFAR-10 dataset, measure $G_n$




(a) VGG-19 (b) Resnet-18 (c) Resnet-34

Figure (4.5) Testing accuracies of the neuron amplified models(red) using $params_2$ where epochs 51-145 are used for amplification using formula 4.1 compared to mean testing accuracies of the original models(blue) with no gradient amplification.




(a) Local normalization (b) Global normalization

Figure (4.6) Testing accuracies of the neuron amplified models(red) as for amplification factors(3,4,5) compared to mean testing accuracies of the original models(blue) with no gradient amplification.

shown in Fig. 4.6. These experiments are performed using both local and global normalization methods with $params_2$ training strategy(epochs 51-145 are amplified). It can be seen that increasing amplification factor slightly decreases the model performance and sometimes has similar accuracy of original models. It does not drastically decrease the performance as seen in random amplification improve the performance of the model.

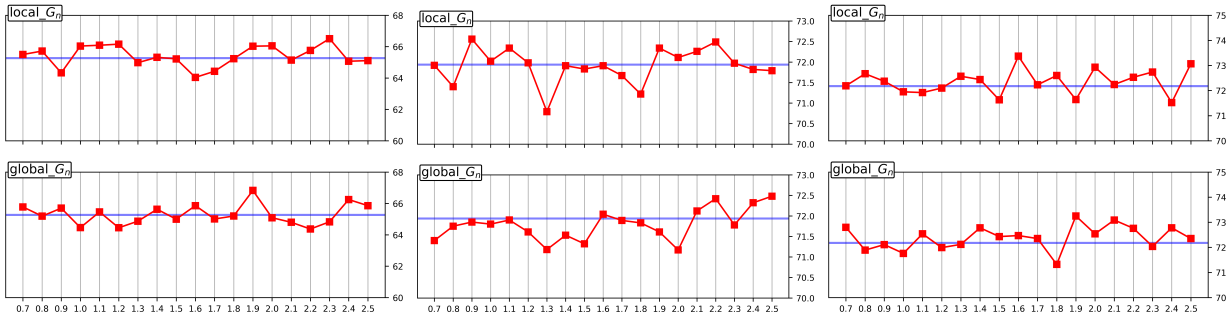
### 4.1.3 Analysis on CIFAR-100 dataset

Fig. 4.7 shows the performance of the models when neuron level amplification is applied during epochs 51-145 on CIFAR-100 dataset. Though models have better performance for are some thresholds than original(no amplification) models, most of the threshold values have neuron amplified models having similar performance and sometimes even with reduced performance than original models. Selection of training strategies also do not have much significance on the model performance. So far, all the analysis is done on experiments with amplification factor as 2.

### 4.1.4 Comparison of running times

Table 4.1 and 4.2 show the mean running times(in minutes) across 10 runs of original models and amplified models for different ratio measures and cases. Training is performed on

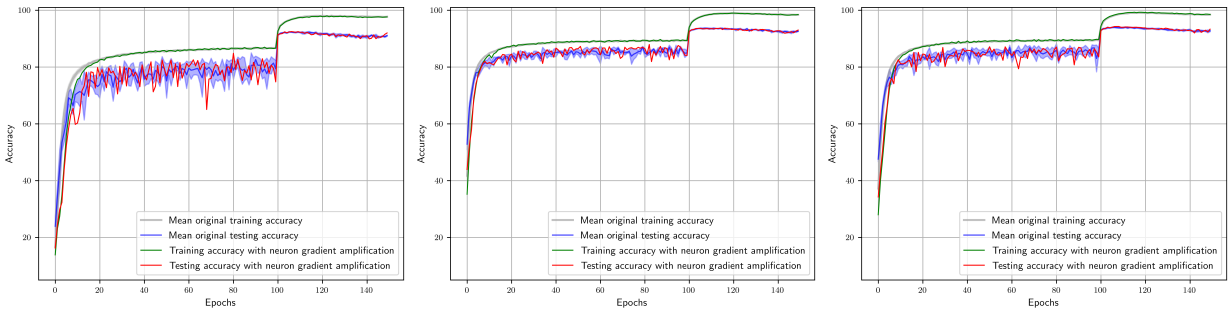
CIFAR-100 dataset, measure $G_n$




(a) VGG-19 (b) Resnet-18 (c) Resnet-34

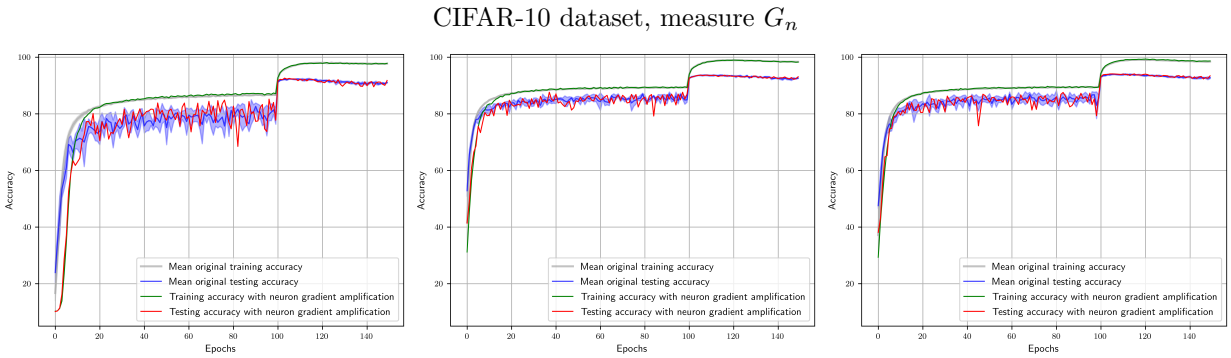
Figure (4.7) Performance of the neuron amplified models(red) using $params_1$ where epochs 51-145 are used for amplification using formula 4.1 compared to mean accuracies of the original models(blue) with no gradient amplification. For each model (a), (b), (c), the upper plot corresponds to the testing accuracies of the models when amplification is done with local-normalization approach and the lower plot corresponds when the layers to global-normalization. Horizontal axis refers to the thresholds applied on the normalized gradient rate ($\hat{G_n}$) and vertical axis correspond to testing accuracies of the model.

CIFAR-10 dataset, measure $G_n$




(a) VGG-19 (b) Resnet-18 (c) Resnet-34

Figure (4.8) (Local Normalization) Performance of the best models with original training(gray), testing(blue) accuracies including their mean accuracies and neuron amplified training(green) and testing(red) accuracies.




(a) VGG-19 (b) Resnet-18 (c) Resnet-34

Figure (4.9) (Global Normalization) Performance of the best models with original training(gray), testing(blue) accuracies including their mean accuracies and neuron amplified training(green) and testing(red) accuracies.

the GSU high performance cluster with NVIDIA V100 GPUs with only our models running on the GPUs with no other user jobs. Clearly, performing amplification does not have computational overhead as it takes around 1-2 min extra times compared to their original models without amplification. In the case of CIFAR-10, resnet-50 models have around 5-7 min more mean computational time compared to the original model.

Table (4.1) Mean running times(in minutes) of models on CIFAR-10 dataset using neuron-level amplification

| Model | *Original* (*min*) | *Ours* with $\hat{G}_n$ amplification (*min*) | |
|---|---|---|---|
| | | global- | local- |
| *VGG_19* | 31.35 ± 0.45 | 31.94 ± 0.52 (**1.87%**) | 32.19 ± 0.4 (**2.68%**) |
| *Resnet_18* | 42.15 ± 0.23 | 42.84 ± 0.53 (**1.66%**) | 42.85 ± 0.57 (**1.67%**) |
| *Resnet_34* | 66.35 ± 0.92 | 67.75 ± 0.91 (**2.11%**) | 67.63 ± 1.1 (**1.93%**) |
| *Resnet_50* | 140.07 ± 1.59 | 150.03 ± 12.07 (**7.11%**) | 146.85 ± 5.02 (**4.84%**) |
| *Resnet_101* | 224.22 ± 3.45 | 228.93 ± 0.91 (**2.1%**) | 229.46 ± 1.75 (**2.34%**) |

### 4.1.5 Best models with neuron amplification

Here, best results all the experiments on neuron based amplification models are compared with the original models without amplification. Fig. 4.8 and 4.9 show the progression of these models for increasing epochs with local and global normalization respectively. These

Table (4.2) Mean running times(in minutes) of models on CIFAR-100 dataset using neuron-level amplification

| Model | *Original* (*min*) | *Ours* with $\hat{G}_n$ amplification (*min*) | |
|---|---|---|---|
| | | global- | local- |
| *VGG_19* | 31.35 ± 0.45 | 32.05 ± 0.48 (**2.23%**) | 31.94 ± 0.42 (**1.88%**) |
| *Resnet_18* | 42.15 ± 0.23 | 42.89 ± 0.48 (**1.76%**) | 42.84 ± 0.48 (**1.65%**) |
| *Resnet_34* | 66.35 ± 0.92 | 67.6 ± 1.04 (**1.89%**) | 67.73 ± 0.83 (**2.08%**) |
| *Resnet_50* | 140.07 ± 1.59 | 141.37 ± 2.4 (**0.93%**) | 140.73 ± 1.71 (**0.47%**) |
| *Resnet_101* | 224.22 ± 3.45 | 229.62 ± 0.57 (**2.41%**) | 227.58 ± 0.71 (**1.5%**) |

best models mostly have similar performance as original models during the initial epochs and have slightly better performance towards the end epochs. Table 4.3 and 4.4 shows the final testing accuracies of original and amplified models after 150 training epochs for CIFAR-10 and CIFAR-100 respectively.

Table (4.3) Final testing accuracies of models with neuron amplification (vs) mean accuracies of corresponding original model on CIFAR-10 dataset

| Model | *Original* | *Ours* with $\hat{G}_n$ amplification | |
|---|---|---|---|
| | | global- | local- |
| *VGG_19* | 91.08% | 91.99% (+**0.91%**) | 92.78% (+**0.7%**) |
| *Resnet_18* | 92.39% | 93.02% (+**0.63%**) | 93.09% (+**0.7%**) |
| *Resnet_34* | 92.71% | 93.33% (+**0.63%**) | 93.41% (+**0.7%**) |
| *Resnet_50* | 91.80% | 92.56% (+**0.76%**) | 92.59% (+**0.79%**) |
| *Resnet_101* | 91.95% | 92.6% (+**0.65%**) | 92.72% (+**0.77%**) |

## 4.2 Summary

We propose a measure to compute the effective gradient direction of a neuron during its training process. Two different ways of normalizing the ratio values are discussed. These normalized measures are used to determine the neurons for gradient amplification. Detailed experiments are performed to analyze each of the normalized measures with varying thresh-

Table (4.4) Performance comparison of models with neuron amplification with CIFAR-100 dataset

| Model | *Original* | *Ours* with $\hat{G_n}$ amplification | |
|---|---|---|---|
| | | global- | local- |
| *VGG*_19 | 65.27% | 66.51% (+**1.24%**) | 66.83% (+**1.56%**) |
| *Resnet*_18 | 71.94% | 72.56% (+**0.62%**) | 72.48% (+**0.54%**) |
| *Resnet*_34 | 72.18% | 73.37% (+**1.19%**) | 73.25% (+**1.07%**) |
| *Resnet*_50 | 72.32% | 73.07% (+**0.75%**) | 72.48% (+**0.16%**) |
| *Resnet*_101 | 72.18% | 73.23% (+**0.23%**) | 73.57% (+**0.57%**) |

olds and amplification factors. It can be seen that neuron level amplification does not have significant improvement over models trained without amplification. This concludes that performing layer level amplification by aggregating the neuron gradient change rates of a layer is effective.

# 5

# DATA INTEGRITY ATTACK DETECTION IN SMART GRID

## 5.1 Background

Cybersecurity plays a vital role in determining the reliability and availability of the entire smart grid infrastructure. Potential intrusions can make the system vulnerable to various types of attacks, each of which may lead to serious outcomes, ranging from the leakage of customer private information to cascade of system failures [68–70]. In the current study, we primarily focus on data integrity attacks in which, measurements of a set of compromised meters are altered by the attacker. Such attacks can mislead the operational metrics at the control center causing the operators to make fatal decisions and disrupting the reliability of measured data. State Vector Estimation (SVE) methods [71] are generally employed to identify such malicious data attacks. In these methods, the state vector, which corresponds to an optimal power system state, is maintained, and a new state vector is reconstructed from measurements. If the difference between the actual and the reconstructed state vector is above a threshold value $\tau$, then the data is considered to be attacked data. The study [72] suggests a way of constructing data integrity attack vectors that fall in the column space of the Jacobian matrix($\boldsymbol{H}$) of the measurements. In those cases, there exist cooperative attacks on meters, known as unobservable attacks, that cannot be identified using state vector estimation or any other known bad data detection techniques. There have been many algorithms proposed to detect such attacks in the literature [73, 74]. Supervised machine learning classification models have also been proposed to identify such attacks [75, 76]. Esmalifalak *et al.* [75] uses a series of measurements as time series data, preprocesses the data using principal component analysis (PCA), and then uses support vector machines (SVM) with Gaussian kernel to identify malicious data attacks. Ozay *et al.* [76] employ a wide array of supervised and semi-supervised machine learning methods on the measurements, and

analyzes the performance of these methods.

Identifying attacked measurements from secure measurements using machine learning techniques is an active research area. In this paper, we use deep learning Feed-Forward Neural Network model to learn and classify malicious data from secured data. This is done by geometrically analyzing and visualizing the measurement space which helps in identifying a distance metric for measurements, that can be used in machine learning algorithms as the cost function, to optimally discriminate secured data from attacked(malicious) data. We employ deep learning models, in this case feed-forward neural networks, to classify secured vs. malicious data. We perform exhaustive performance analysis of the classification models for a range of attacked meters for multiple IEEE bus test systems and also compare our results with other physical and machine learning models. This chapter is based on the publication [64] and further details can be found therein.

### 5.1.1 Preliminaries

### 5.1.2 State Estimation

State Estimation is the process of identifying the current state of the system from measurements gathered across various meters. Monitoring such information is necessary for the reliability of the system, as the operators at the control center use this information to re-dispatch power, call on operating/contingency reserves, re-balance control areas, and for auditing and settlement [74]. In this process, the control center collects real time measurements $\boldsymbol{z}$ across various meters of the smart grid system and combines the network topology and parameter information, to calculate the real time estimates of the unknown system state variables $\boldsymbol{x}$. Let $\boldsymbol{z} = (z_1, z_2, ..., z_m)^T$ denote the meter measurements generated using bus power and branch power flow measurements and $\boldsymbol{x} = (x_1, x_2, ..., x_n)^T$ denote the system state variables computed using voltage phases at all buses, where $m$ is the number of meters and $n$ is the number of unknown state variables, and $m \geq n$. Let $\boldsymbol{e} = (e_1, e_2, ..., e_m)^T$ denote the measurement errors with Guassian noise with zero mean and a covariance matrix $\sigma^2 I$. The

measurement model for DC power flow is given as follows:

$$\boldsymbol{z = Hx + e}$$

where $\boldsymbol{H}$ is an $m \times n$ full rank Jacobian matrix of the measurement model. The matrix $\boldsymbol{H_{m \times n}}$ signifies the relationship between the vector $\boldsymbol{z_{m \times 1}}$ of measurements consisting of $m$ meters readings, and the state vector $\boldsymbol{x_{n \times 1}}$ consisting of $n$ state variables $x_1, ..., x_n$. From the vector of measurements, the estimated state vector $\hat{\boldsymbol{x}}$ is computed using

$$\hat{\boldsymbol{x}} = (\boldsymbol{H^T W H})^{-1} \boldsymbol{H^T W z} \tag{5.1}$$

where $\boldsymbol{W}$ is a diagonal matrix whose elements are reciprocals of the variances of meter errors.

### 5.1.3 Bad Data Detection

Bad measurements may be introduced due to some faulty transmission lines, meter failures, or due to malicious attacks. Estimated state vector of normal measurements is close to the actual state vector, while deviates from it for malicious measurements. The difference between the observed measurements $\boldsymbol{z}$ and the computed measurements using $\hat{\boldsymbol{x}}$ i.e., $\hat{\boldsymbol{z}} = \boldsymbol{H}\hat{\boldsymbol{x}}$ is known as the measurement residual, $\boldsymbol{z} - \hat{\boldsymbol{z}}$ . $L2$-Norm of this residual $\|\boldsymbol{z} - \boldsymbol{H}\hat{\boldsymbol{x}}\|_2$ is generally used to detect bad measurements using a threshold value $\tau$. If the measurement residue exceeds this threshold value i.e., $\|\boldsymbol{z} - \boldsymbol{H}\hat{\boldsymbol{x}}\|_2 > \tau$, the measurements are considered to be compromised; otherwise they are considered as secured measurements.

### 5.1.4 Unobservable Attacks

Injecting malicious data involves modifying the meter readings so that the actual measurements are augmented with an attack vector $\boldsymbol{a} = (a_1, a_2, ..., a_m)^T$, that have non-zero elements for compromised meters. The compromised vector of measurements can then be

constructed as:

$$\hat{\boldsymbol{z}} = \boldsymbol{z} + \boldsymbol{a}$$

The attacker can be assumed to have access to the matrix $\boldsymbol{H_{m \times n}}$, in addition to some or all of the meters. Let $\mathcal{A} = \{t_1, t_2, ..., t_k\}$, where $t_i \in \{1, 2, .., m\}$, be the set of meters that are compromised and $\tilde{\mathcal{A}}$ be the uncompromised meters. When the compromised measurements are coherent with the physical power flow constraints of the system, such attacks are called as *unobservable* attacks. These attacks cannot be detected by State Estimation methods or any other bad data detection algorithms. Such an unobservable attack vector $\boldsymbol{a} = (a_1, a_2, ..., a_m)^T$ can be generated using the following condition:

$$\boldsymbol{a} = \boldsymbol{Hc} \tag{5.2}$$

where $\boldsymbol{c}$ is a sparse vector using guassian distribution, and $a_i \neq 0$ for $i \in \mathcal{A}$ and $a_i = 0$ for $i \notin \mathcal{A}$. The properties and constraints for generating unobservable data integrity attacks are briefly described in [72].

The compromised measurements generated after adding the above attack vector is as follows:

$$\hat{\boldsymbol{z}} = \boldsymbol{H}(\boldsymbol{x} + \boldsymbol{c}) + \boldsymbol{e}$$

where the injected vector $\boldsymbol{c}$ is the residue error introduced into the measurements, which goes undetected.

There are two types of attacks possible for centralized data integrity attacks, namely, random false data injection attacks and targeted false injected attacks. In the former type of attacks, the attacker can corrupt measurements from any of the meters to make the attack undetectable in state estimation, whereas in the latter, the attacker has access to only a specific set of meters. In this paper, we use LASSO optimization methods [77] to generate attack vectors.

## 5.2 Proposed Problem Formulation

Attacked measurements cannot be identified using state estimation methods (SVE) when the attack vectors are in the column space of $\boldsymbol{H}$ as in equation (5.2). Such vectors can always be constructed when the number of attacked meters is more than $m-n+1$ [72]. These generated attack vectors are closely located around the normal values as shown in Fig.5.1(a), for IEEE 57-bus test system, which generally holds good for any system. All the meter measurements of vector $\boldsymbol{z} = (z_1, z_2, ..., z_m)^T$ are plotted using $(i,\ z_i)$ for $i \in 1, 2, .., m$. For case-57 bus system, $m$=137. Here, we only show a part of the plot for values of $i \in \{13, .., 62\}$ to clearly demonstrate that the malicious and the secure measurements are very closely placed in vector space, separated by very short distances, as shown in the rectangular box. The points on the histogram connected to the rectangle correspond to the zoomed regions showing how these points are separated by some distance. Clearly, the compromised measurements are spaced very closely to secure measurements, separated by short distances. These inter-measurement distances can be used in machine learning models to separate attacked and unattacked measurements in space, and to identify the discriminating boundaries that help classify new measurements efficiently. Since the distances between the measurements are usually small, firstly, the measurements are transformed to logarithmic space, and then the distances are computed.

Formally, let $\boldsymbol{z} = (z_1, z_2, ..., z_m)^T$ denote the actual meter measurements, and $\boldsymbol{a^p} = (a_1^p, a_2^p, ..., a_m^p)^T$ and $\boldsymbol{a^q} = (a_1^q, a_2^q, ..., a_m^q)^T$ be the potential attack vectors generated for $\boldsymbol{z}$. Let $\hat{\boldsymbol{z}}^{\boldsymbol{p}} = \boldsymbol{z} + \boldsymbol{a^p}$ and $\hat{\boldsymbol{z}}^{\boldsymbol{q}} = \boldsymbol{z} + \boldsymbol{a^q}$. Clearly, $\hat{\boldsymbol{z}}^{\boldsymbol{p}} = (\hat{z}_1^p, \hat{z}_2^p, ..., \hat{z}_m^p)^T$, where $\hat{z}_i^p = z_i + a_i^p$. Let $M_i^p, M_i^q$ represent the meter $i$ of measurement vector $\hat{\boldsymbol{z}}^{\boldsymbol{p}}$ , $\hat{\boldsymbol{z}}^{\boldsymbol{q}}$ respectively. The problem can be formulated as below:

$$d(\hat{z}_i^p, \hat{z}_i^q) = \begin{cases} \|log_(|a_i^p|) - log(|a_i^q|)\|_2, & if M_i^p, M_i^q \in \mathcal{A} \\ \|log(|a_i^p|)\|_2, & if M_i^p \in \mathcal{A}, M_i^q \notin \mathcal{A} \\ 0, & \text{if } M_i^p, M_i^q \notin \mathcal{A} \end{cases}$$

where $d(.,.)$ is the distance measure between the two points. Since the meters of the measurement vectors can be attacked independently, $M_i^p$ and/or $M_i^q$ can be attacked. The above formulation handles all these cases, and this distance measure can be used in any machine learning algorithms as the loss or optimization function to train the models so as to effectively separate secure vs. attacked measurements in vector space and to detect attacks on new measurements. We use the above formulation in our deep learning model as the loss function and optimize it to build a robust model for classifying normal vs. malicious measurements.

Given $\mathcal{S} = \{s_i\}_{i=1}^m$, the set of measurements from all the meters, and the corresponding label set $\mathcal{Y} = \{y_i\}_{i=1}^m$, where $s_i \in \mathbb{R}$, $y_i \in \{0, 1\}$. The value of $y_i$ corresponding to the measurement $s_i$ is defined as follows:

$$y_i = \begin{cases} 1, & \text{if the measurement } s_i \text{ is compromised} \\ 0, & \text{otherwise} \end{cases}$$

The input data $(s_i, y_i) \in \mathcal{S} \times \mathcal{Y}$ is assumed to be independent and identically distributed to a joint distribution $\mathcal{P}$ [78]. The sampling from $\mathcal{P}$ is done independently and identically, as per the assumptions in [76]. The aim is to determine a learning model defined as a function mapping $f : \mathcal{S} \to \mathcal{Y}$, which determines the relationships between $\mathcal{S}$ and $\mathcal{Y}$ using the distance measure $d(.,.)$ and separates the attacked and secure measurements.

## 5.3 Deep Learning For Attack Detection

In this section, we briefly describe the design of the software, the architecture of the deep learning model employed to build a classification model to identify secure data and attacked data.

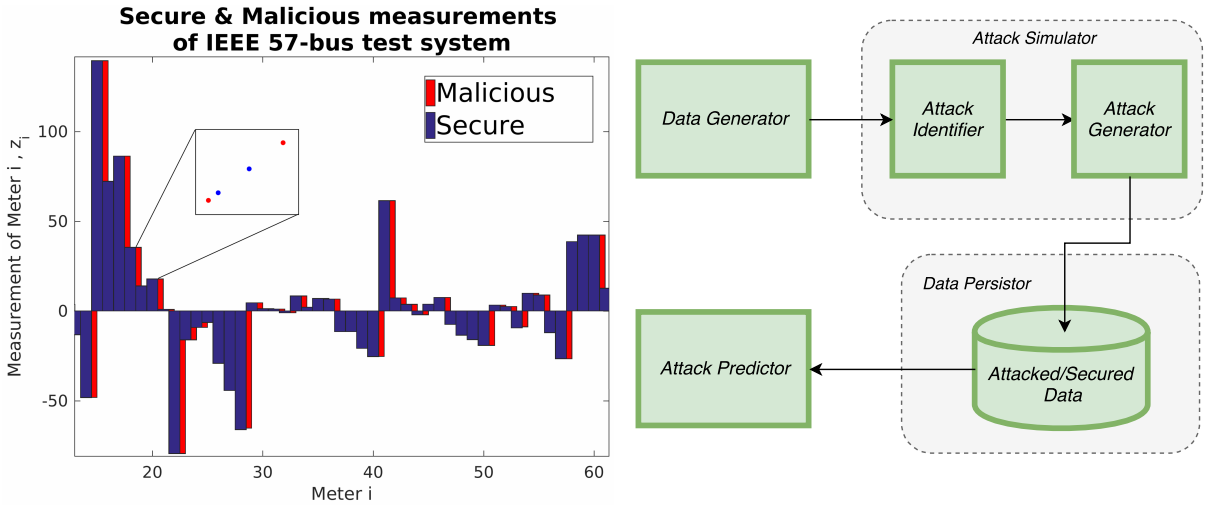
### 5.3.1 Workflow Design

The design of our tool is shown in the Fig. 5.1(b). *Data Generator* module is responsible for generating the metered data of the smart grids. Any software that simulates power

grid models to generate meter readings (or measurements) at different time intervals or the actual meter readings available from power grid websites like PJM [79] can be used in the *Data Generator*. Measurements of all the meters for each timestamp are stored into a matrix. This matrix data is passed to the *Attack Simulator* which has two modules *Attack Identifier* and *Attack Generator*. *Attack Identifier* determines timestamps for the attack, this determination is done by randomly selecting the timestamps. Once a timestamp is considered for the attack, then all the meter readings corresponding to that timestamp are passed to the *Attack Generator*. An attack vector is generated for the given measurement vector and the attacked measurements along with the attacked meters are passed to the *Data Persistor*. Here, the module looks the attacked meters passed from the *Attack Generator*. If any of the meters are passed as attacked meters, then it labels the corresponding measurements with label 1 otherwise, the measurements are labeled as 0. This measurement data along with the labels are stored and also passed to the *Attack Predictor*. This module is reads the labeled measurements and builds a classification model to predict if an unseen measurements are secured or attacked. In our work, we use deep learning models to perform this classification.*Attack Predictor* module can also be independently executed using the data stored by the *Data Persistor*.

### 5.3.2 Deep Learning Model Architecture

There are many neural network architectures such as Convolution deep neural networks (CNNs), Recurrent neural networks (RNNs), and LSTM networks which have outperformed in image recognition, object identification, natural language processing, next sequence prediction. However, our data consists of measurements and therefore a simple feed forward network architecture is employed. In our classification model, the neurons on the input layer accept the actual meter measurements, transform them using activation functions, and pass it on to its successive hidden layers, which in turn transform the data, and finally pass it to the output layer which transforms and outputs a binary value, 0 or 1, classifying the data as secure or attacked respectively.



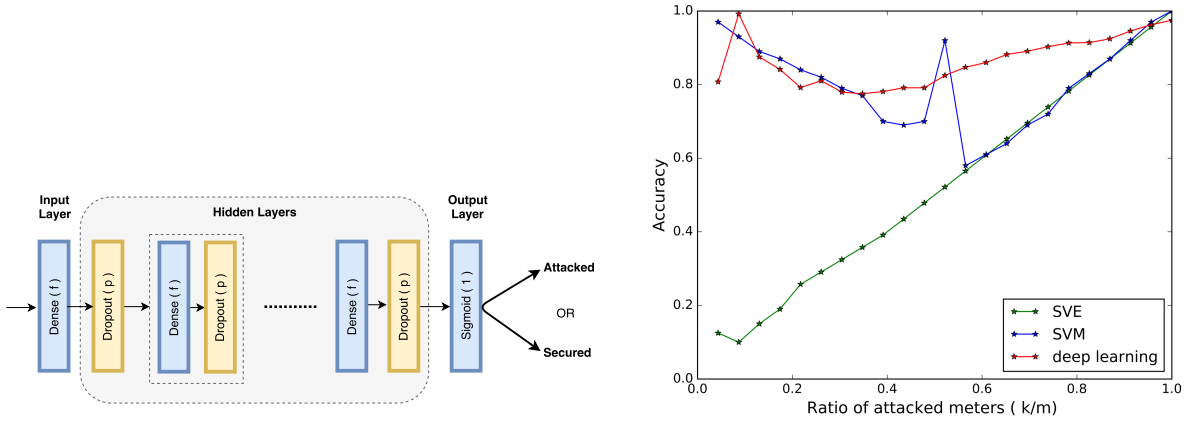

(a) Measurement plot (b) Architecture

Figure (5.1) (A)Secure (vs) malicious measurements of IEEE 57-bus test system. (B) Workflow design of the smart grid data generation and prediction model.

The feed-forward architecture of our neural network is shown in the Fig. 5.2(a). The input layer which is fully connected to the subsequent layer and accepts the input data. The output layer is fully connected to its preceding layer and gives us the classification result. Each of the hidden layers is fully connected to its subsequent layer, however dropout method is used to drop a few neurons and network connections to prevent the network from overfitting. Batch normalization is also performed during training the model and overfitting of the model is prevented using dropout [80] for each of the hidden layers. The number of neurons in each layer is controlled by a parameter $K$ and the dropout factor for each of the layer is determined using the parameter $p$.

## 5.4 Implementation

*Data Generator* module is implemented in Matlab and the simulated measurements for different power grid bus architectures are generated using MATPOWER toolbox [81]. We use MATPOWER to simulate IEEE 9-bus, 14-bus, 30-bus and 57-bus test systems. *Attack Identifier* randomly determines if the measurements obtained from the *Data Generator* should be attacked or not. If it determines the measurements should not be attacked, then the




(a) Architecture (b) Accuracy

Figure (5.2) (A) Architecture of deep learning model employed for detecting attacks. (B) Accuracy for IEEE-57 bus system

measurements are sent to the *Data Persistor* module and the label for these measurements are marked as 0. Otherwise, the attack vector for the given measurements is generated using false data injection attacks. The *Attack Generator* module is implemented using by Algorithm 1 of [77]. This module generates the attack vector of different sparseness based on the ratio $\kappa/m \in [0, 1]$. In our implementation, we generate explore 22 ratio values equally spaced between (0,1). The ratio parameter value multiplied by the number of meters for the test system determines that number of meters to be attacked. This value is added as one of the stopping constraints of Algorithm 1 of [77] so that it corresponds to the sparseness of the attack vector. These generated unobservable attack vectors are added to the original measurements to get the corrupted or attacked measurements. These attacked measurements are passed to the *Data Persistor* module and are stored with the label 1. *Data Persistor* module stores these measurements along with the corresponding labels in the file system as CSV files. These CSV files along with their labels are used by the *Attack Predictor* module to perform classification. Our deep learning models are implemented using Keras [82], a high-level API for neural networks that works on top on TensorFlow [83] and Theano. This API provides a simple interface to implement complex neural network models. Our code

Table (5.1) Calculation of performance measures

| | Attacked | Secure |
|---|---|---|
| Predicted as Attacked | $tp$ | $fp$ |
| Predicted as Secure | $fn$ | $tn$ |

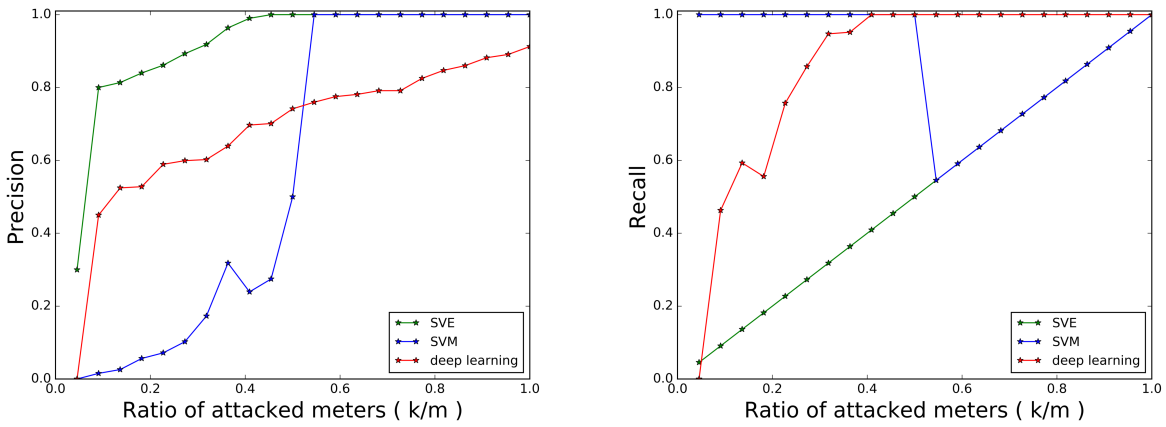
implements the architecture shown in Fig.5.2(a) to train a feed forward neural network with one input layer, one output layer, and multiple hidden layers. Weight initialization methods, activation functions, loss functions, optimizer methods, and metrics which need to be improved, can be passed as method parameters. In our code, we train the model to achieve best accuracy metric using our distance formulation as the loss function. The weights in the network are regulated so as to achieve the best possible accuracy.

### 5.4.1 Sampling and Metrics

Stratified 10-fold cross validation is used for training and testing the model. Accuracy, precision, and recall are the metrics used to study the performance of the model. Accuracy (Acc) is defined as the proportion of test cases correctly classified as either attacked or secure. Precision (Prec) measures the fraction of the test cases which are classified as attacked, have truly been attacked. Recall (Rec) measures the fraction of the test cases that are correctly predicted as attacked. To calculate these metrics, we measure true positives ($tp$), true negatives ($tn$), false positives ($fp$) and false negatives ($fn$), as shown in Table 5.1.

Acc=$\frac{tp+tn}{tp+tn+fp+fn}$ , Prec=$\frac{tp}{tp+fp}$ , Rec=$\frac{tp}{tp+fn}$

In our models, the input data $(s_i, y_i) \in \mathcal{S} \times \mathcal{Y}$ is sampled and used for training feed-forward neural networks. Each measurement $s_i$ is fed as input to the model and the output layer with a single neuron results in either 0 or 1, to represent if the data is secure or attacked respectively. Problem formulation defined in Section 5.2 is employed into the cost function and the model is trained so as to optimize the cost function and reduce the prediction errors.




(a) (b)

Figure (5.3) Performance comparison among state vector estimation (SVE), SVM (Linear) and deep learning models for IEEE 57-bus test system. (A) Precision (B) Recall

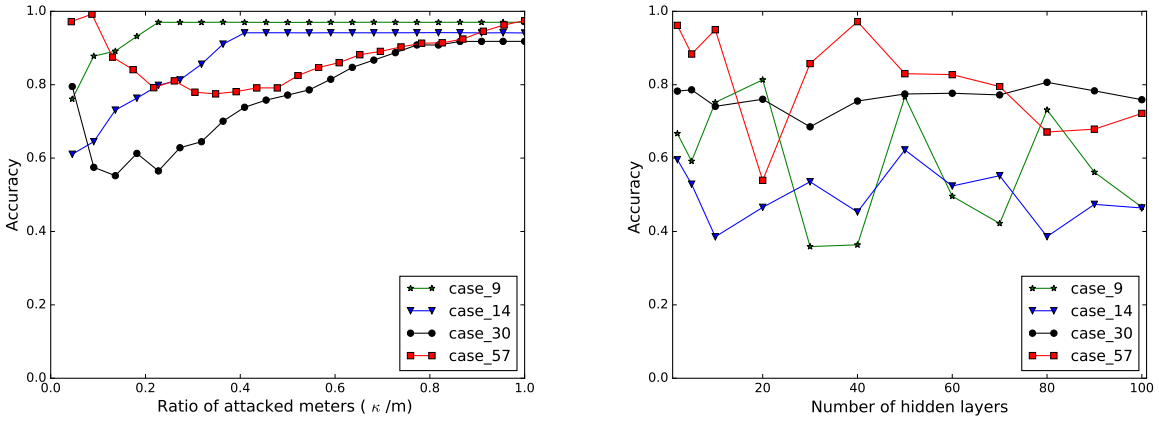
## 5.5 Experimental Results

The classification models are analyzed for IEEE 9-bus, 14-bus, 30-bus, and 57-bus test systems in our experiments. MATPOWER toolbox [81] is used to compute the measurement Jacobian matrices $\boldsymbol{H}$. The operating state of the system $\boldsymbol{x}$ is obtained from MATPOWER, and then the measurements for $\boldsymbol{z}$ are computed. For each of the measurement vectors, we independently determine if the measurements need to be attacked. Following this, the corresponding attacked measurements using random and targeted false injection attacks are generated [72, 77]. Data is labeled '1' if the measurements are attacked, or '0' otherwise. In the experiments, we assume that the attacker has access to $\kappa$ meters, and these meters are randomly chosen to generate a $\kappa$-sparse attack vector. These attack vectors have non-zero values with the same mean and variance as $\boldsymbol{z}$, and follow a Gaussian distribution. We evaluate the behavior of the models to classify both observable and unobservable attacks for different values of $\kappa/m \in [0, 1]$. Specifically, we analyze the performance of the model for $\kappa \geq m - n + 1$, where unobservable attacks are generated, and the attack vectors are not observable by SVE [72]. Otherwise, the generated attacks are observable.

Feed-forward neural networks are implemented using Keras API [82] in python with

Tensorflow as the backend library. There are multiple parameters to learn in the deep learning models, such as the number of hidden layers, the number of neurons in each of these layers, the type of activation functions for neurons, the dropout factor and, the loss functions. In our models, we integrate the distance metric in the problem formulation as the loss function, and use Exponential Linear Units (elu) [84] as the activation function for all the neuronal units of the hidden layers. Sigmoid is used as the activation function for the output layer so that the output result can be directly related to the class label. The number of neurons in the input and output layers are fixed as it depends on the dimension of the input data and the type of output. To determine the appropriate number of hidden layers, the model is parameterized to have upto 100 hidden layers, and the number of neurons (f) in each layer is taken from the set $\{0, 50, 100, .., 450, 500\}$. Dropout factors (p) are chosen from the set $\{0.25, 0.50\}$. Scikit [85] library from python is used to perform grid search through the parameter space, and determine the best possible parameters for the model. We also train Linear SVM models using scikit library in python for comparison studies.

Fig.5.2(b) shows the accuracies of SVE, SVM (Linear), and deep learning models for IEEE 57-bus test system. Accuracies are measured using a stratified 10-fold cross validation method. It can be seen that SVM performs comparable to the feed-forward deep neural network model till the ratio $\kappa/m$ reaches 0.5, after which, the feed-forward network (deep learning) model performs better. SVE (State Vector Estimation) performance increases linearly with the number of meters, but it also increases the true negatives (number of outputs predicted as attacked, but are actually unattacked). This can be inferred from the precision and recall comparison figures shown in Fig. 5.3(a), 5.3(b). Linear SVM models have better recall values when less than half of the meters are attacked, but have very low precision values due to the presence of false positives and vice versa, when more than half of meters are attacked. SVE models have good precision performance, but have less recall performance which shows that not all the attacked meters are identified correctly. Deep learning models have less precision performance when compared to SVE, but have better recall performance than SVE. They also have better accuracy performance when compared



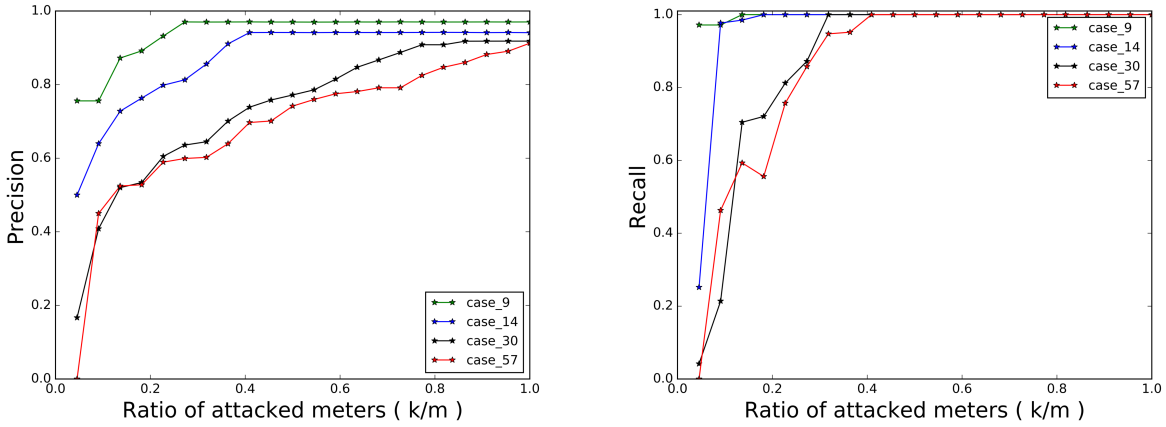

(a) (b)

Figure (5.4) Accuracies of all the models with the increase in (a) number of attacked meters (b) number of layers in the model

to SVE and SVM models. Deep learning models trained with gradient amplification improves the accuracies of the original deep learning models by 1-2% for all the IEEE bus systems.

The performance of neural network model slightly increases with an increase in the number of layers, after which it stabilizes or sometimes decreases after a maximum value. One of the reasons for this reduction in accuracy with the increase in number of layers is due to overfitting of the data with the increase in number of layers. Fig. 5.4(b) shows the accuracy of models with the increase in number of layers. Clearly, accuracy of some IEEE bus test systems reach highest accuracy with almost two layers which some models require more layers. In Fig. 5.4(b), we observe that IEEE 9-bus, 14-bus test systems achieve their maximum accuracy around 50 hidden layers, IEEE 30-bus test system achieves similar highest accuracies around 2 or 90 hidden layers and IEEE 57-bus test system achieves it around 2 or 40 hidden layers.

The accuracies of the models vary based on the number of attack meters and also the other hyperparameters. We show metrics for neural network models with the best performance values of all trained models for IEEE 9-bus, 14-bus, 30-bus, and 57-bus test systems. The performance charts can be seen in Fig. 5.4(a), 5.5(a), 5.5(b). It is observed




(a) (b)

Figure (5.5) Performance (a) precision (b) recall of the models for different IEEE bus systems

that all the models start to converge to high accuracy when more than 55% of the meters are attacked. IEEE 9-bus test system achieves around 90% accuracies with just 2 hidden layers. The number of hidden layers and neurons increase with the number of buses in the IEEE test systems. Clearly, feed-forward neural networks achieve more than 90% accuracies when the ratio of the attacked meters is greater than 0.3 for IEEE 9-bus, 11-bus test systems, and is greater than 0.7 for IEEE 30-bus, 57-bus test systems.

To analyze the computational time for each of the models with the increase in the number of layers, we observe in Fig. 5.6 that the average training time increases with the increase in the number of layers. The horizontal axis in the Fig. 5.6 represents the number of hidden layers in the model and the vertical axis represents the average time taken for the model to train the across the parameter space. The interesting observation is that as the complexity of the IEEE bus test system increases, the average training time is less compared to the simplest bus system. That is, as the number of layers increases, IEEE 9-bus system has more average training times compared to IEEE 14-bus test system which in turn has higher average training times compare to IEEE 30-bus test system. IEEE 30-bus and 57-bus has comparable (close) training times which is less than IEEE-9, IEEE 14-bus systems.

Since we experiment with multiple IEEE bus test systems, namely 9-bus, 14-bus, 30-

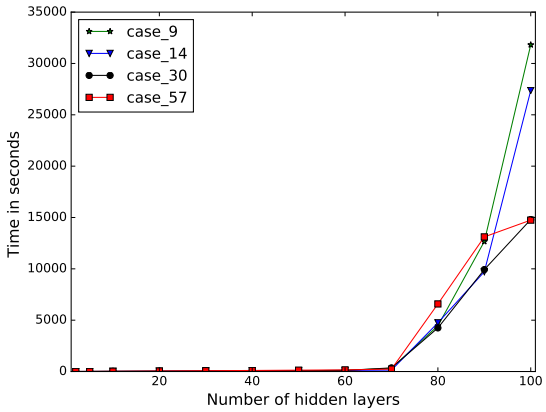



Figure (5.6) Average training times of deep learning models for IEEE 9-bus, 14-bus, 30-bus, and 57-bus test systems, where there are 10 attacked meters. The horizontal axis represents the number of hidden layers in the model and the vertical axis represents the average time taken for the model to train the across the parameter space.

bus, and 57-bus systems, to achieve fast computation results, we perform the data generation for all the bus systems in parallel using multiple processes. For each of the test systems, we analyze the scenario when different number of meters attacked and analyze their classification performance. This means for each of the datasets corresponding to different ratio of attacked meters, a model needs to trained and tested. To accelerate the training process for all the models, we train these models in parallel on a server and persist the optimal network models.

## 5.6 Summary & Future Work

The data integrity attack detection problem in smart grids is redefined as a machine learning problem, and a feed-forward neural network model is employed for the classification of attacked data and secured data. The performance of this model is analyzed and compared with State Vector Estimation (SVE) and supervised machine learning algorithms for a range of attacked meters. This analysis shows that feed-forward neural networks can detect attacks with better performance than attack detection algorithms that employ state vector estimation methods for centralized data integrity attacks. Also, the performance of feed-forward deep neural networks against various IEEE-bus test systems is compared. It is clearly seen that feed-forward neural networks perform better than State Vector Estimation methods

and have comparable (often better) performance with other supervised machine learning algorithms. In future work, we would like to improve our results furture using building hybrid models using evolutionary algorithms as in [86–91]. We would also like to work on the real-time measurement data from PJM [79] and employ other deep learning models such as recurrent networks and LSTMs [92] which are known to work better for sensor data, time series and high-dimensional data to improve the performance of the classification of secured and attacked data.

# 6

# CLASSIFICATION OF HCV INFECTIONS THROUGH SEQUENCE IMAGE NORMALIZATION

## 6.1 Background

Analysis of heterogeneous viral populations is one of the most challenging bioinformatics tasks owing both to the complexity of the underlying algorithmic problems and features and sheer amount of data [93, 94]. These challenges became especially complicated in the recent decade with the advent of high-throughput sequencing (*HTS*), which has now become a major tool for viral research, allowing to sample viral populations at unprecedented depth [95–101]. Modern computational virology continues mostly to rely on classical approaches, which include sequence analysis, phylogenetics/phylodynamics and structural bioinformatics [94,102]. In the recent years, these approaches started to be complemented with the network analysis [103–105].

In order to employ machine learning for viral studies, quasispecies populations should be transformed into feature vectors from a multidimensional euclidean space. Several encoding schemes have been used in the literature for transforming biomedical data into numerical data for machine learning [106]. However, the existing methods face significant challenges when applied to viral genomic data. These challenges are associated with extremely high heterogeneity of intra-host viral populations, sequencing errors and sampling biases, robustness to noise and difficulty of selection of relevant sets of features. This chapter is based on the publications [62, 63] and further details can be found therein.

## 6.2 Challenges in using sequence data for machine learning

There are several challenges that currently impede applications of machine learning and deep learning methods to viral studies. These challenges could be thematically classified as

follows:

### 6.2.1 Challenges associated with technological limitations.

High-throughput sequencing technologies are prone to errors and biases, which may significantly affect viral data. Indeed, frequencies of minor viral variants are often comparable with the level of sequencing noise; however, such variants should not be simply discarded based on some frequency threshold, since often they are the ones responsible for transmission, immune escape or therapy failure [107–112]. Presence of sequencing errors introduces noise to data and produces outlier viral variants, which negatively affect the quality and accuracy of machine learning classifiers.

Another important problem is sampling and sequencing bias resulting in significant irregularities in the number and length of viral sequences from different infected individuals. If classifiers capture these artificial differences as significant associations, it may result in overfitting and decline of accuracy. Thus, application of machine learning to heterogeneous viral population data should be preceded by a preprocessing step to eliminate these irregularities via normalization procedure. However, selection of an appropriate normalization approach is challenging. For instance, if we use text classification techniques for preprocessing, difference in the number of sequences among different files needs to be controlled either by truncation or padding. This preprocessing, however, causes data loss (in case of truncation) or introduces irrelevant data (in case of padding). An optimal preprocessing method should not introduce such issues.

### 6.2.2 Challenges associated with feature selection and feature extraction.

Before applying machine learning methods to classification of heterogeneous viral populations, genomic data should be mapped into the euclidian space $\mathbb{R}^n$. It is usually achieved by identifying numerical features that are relevant to the problem under consideration. They can include various diversity measures [113], population genetic parameters [114], physicochemical properties [102] and other parameters specifically tailored to particular problems.

These features are generally identified in consultation with domain experts. Selection of the most relevant features is daunting and resource-consuming. A role of feature selection in determining classification performance is paramount. Selection of a limited number of features from certain domains inevitably results in loss of information, while increase of feature space dimensionality increases risk of overfitting and compromises the algorithm's scalability.

A optimal feature selection method should be able to capture the entire population structure using a relatively simple and easily constructable data representation. Furthermore, it should use a standard universal data format, which has a fixed number of features and is applicable to different problems. Since genomic data are essentially a textual information, it is tempting to utilize well-developed machinery from the text classification domain [115, 116] for the purpose of construction of such representation. Viral populations could be mapped to a euclidian spase using word2vec approaches [117], and classified using various available deep learning models [115,116]. However, application of text processing approaches to viral research could be impeded by several factors. Since they are based on deep neural network models with large numbers of hyperparameters, it requires large annotated datasets to train these models. However, in molecular epidemiology, the amount of available training data is usually limited in comparison with the text processing domain. The datasets of several hundred intra-host viral populations analyzed in this paper are typical in this context. Although, word2vec or document embedding methods can be directly employed, it is challenging to train a model to get a higher classification performance. Furthermore, since viral haplotypes are unique, the trained model could overfit the data.

### 6.2.3 Challenges associated with data comparison.

Clustering of intra-host viral populations requires an inter-population distance measure, which takes into account complex population structures. It has been shown that among simple alignment-based population distance measures, the minimal distance between population variants allows to achieve the highest clustering accuracy [118]. However, this measure is sensitive to noise and presence of outliers, and does not take into account the whole pop-

ulation structure. Recently, several simulation-based and network-based distance measures have been proposed [104, 105], which overcome above-mentioned limitations at the cost of lesser scalability. Thus, the universal, accurate and efficiently computable inter-population distance measure, which takes into account complex population structures still has to be developed.

### 6.3 Sequence Image Normalization: Proposed Preprocessing Method

We transform sequence data into an image by the preprocessing method further referred to as Sequence Image Normalization. We assume that sequences are aligned and ordered by their counts, with sequences of the same counts being sorted lexicographically. Next, each symbol $l \in \{'A','C','T','G','-'\}$ is associated with a particular color thus transforming the sequence alignment into an image. Finally, the images corresponding to different infected hosts are normalized by transforming them into fixed size images. The colors to represent nucleotides are selected from the set of colors of higher variation in order to simplify identification of discriminative features characterizing particular intra-host populations. Fig.6.1 demonstrates an example of sequence image normalization output. Normalized images thus allow to captures entire viral population structure using a single data representation independent of the number of sequences and with minimum loss of existing data or introduction of artificial data.

Raw pixel data of generated images are used as features to train machine learning models for the consecutive analysis, as demonstrated in Fig. 6.2. The number of features depends on the image resolution: each image of the resolution $x \times y$ corresponds to $x \times y \times 3$ feature vector, with each pixel having 3 RGB components. In our experiments, sequence datasets have been analyzed for different resolutions ranging from $50 \times 50$ to $550 \times 550$ with the step size of 50 in each dimension. Results were generated using resolution $480 \times 480$ at which both models performed most accurately.

Thia new approach allows to utilize a well-developed machine learning methodology from the domain of image processing in genomic analysis. The proposed scheme provides

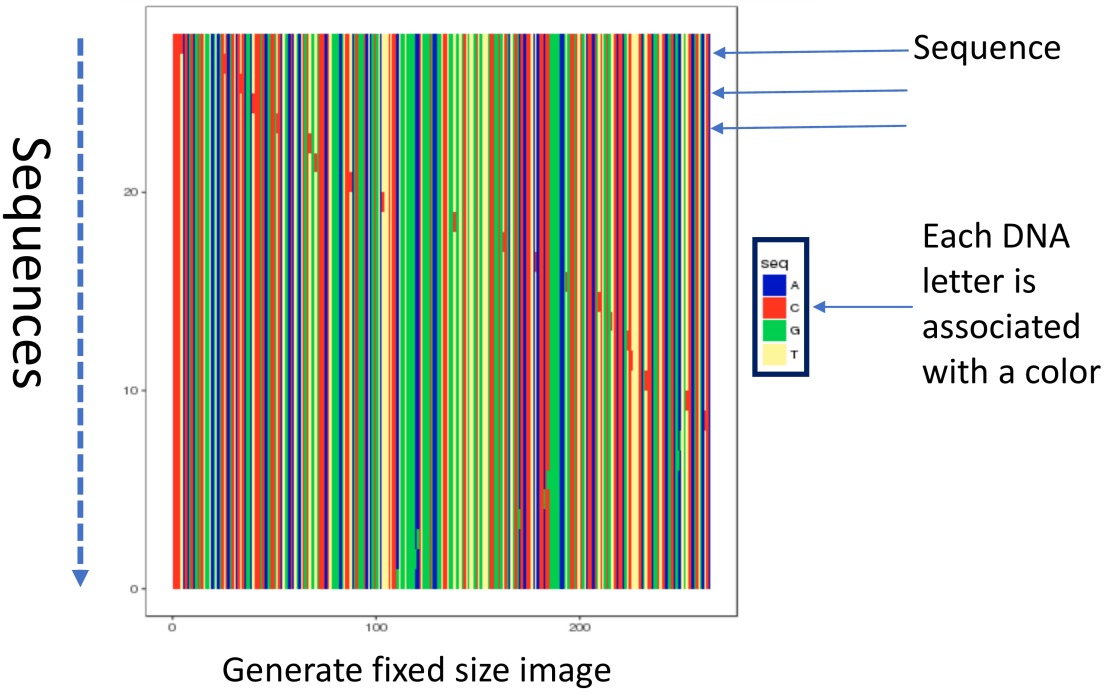



Figure (6.1) Generation of fixed size image

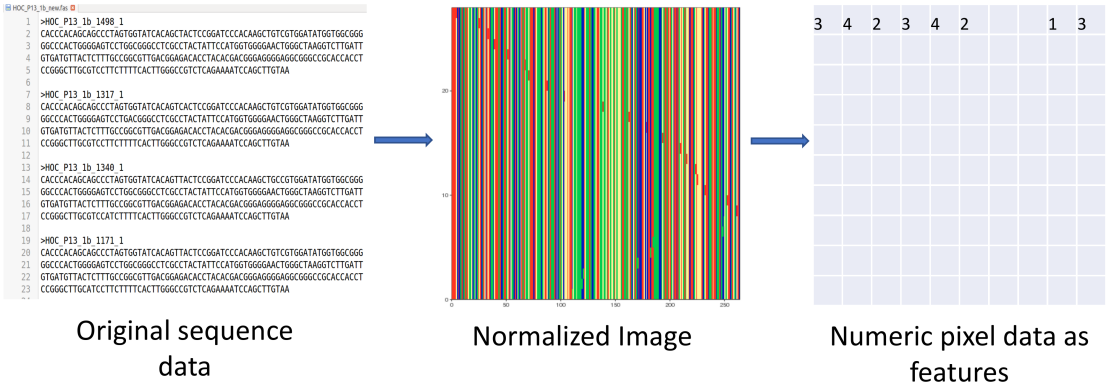



Figure (6.2) Pipeline of sequence image normalization of a fasta file

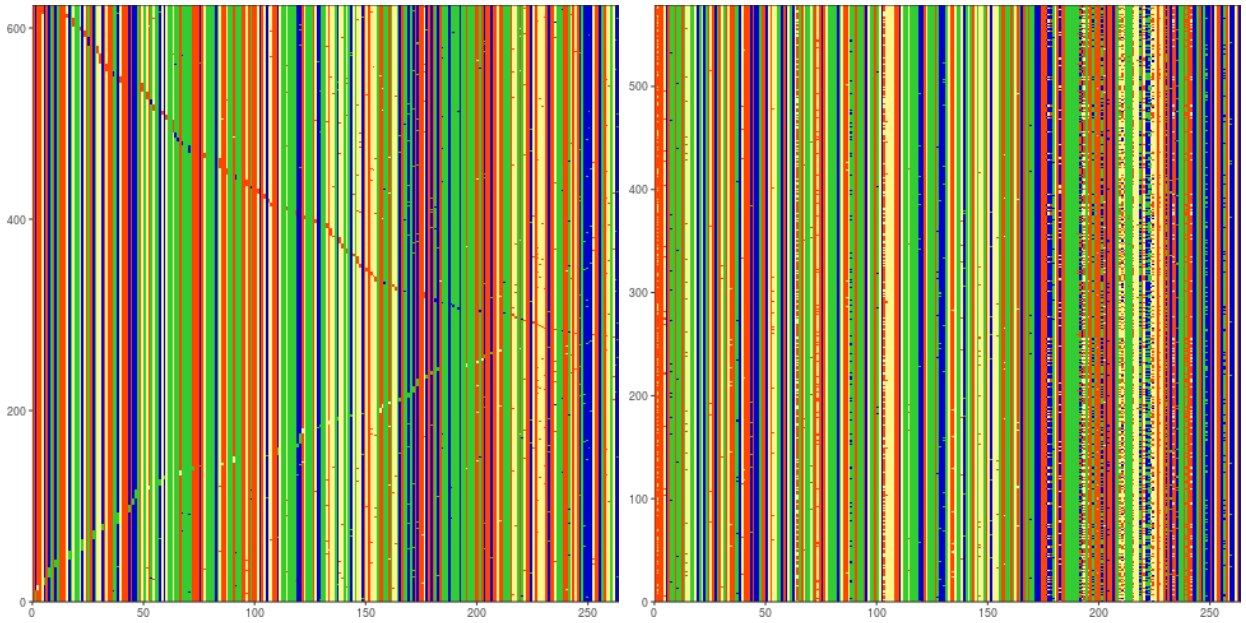
the data structure for the representation of intra-host population structure which is compact, easily adjustable, robust to technological noise and sampling bias, preserve structural properties of populations and can be used for a variety of classification problems, where machine learning is applicable.

## 6.4 Validation

We validated our approach by applying image processing techniques to two important molecular epidemiology problems. The first problem is the HCV infection staging, i.e. distinguishing between recent and chronic infections using viral sequences sampled by next-generation sequencing (NGS). The second problem is detection of outbreaks using NGS data. In molecular epidemiology, it is common to utilize the observation that viral populations from the same outbreak are genetically related. Thus, measures of genetic relatedness could be used as a predictor for epidemiological relatedness [119–121]. In other words, this problem could be considered as the problem of clustering of intra-host viral populations.

### 6.4.1 Classification of HCV infection

**Data** Intra-host HCV populations sampled by sequencing of a highly heterogeneous genomic region (HVR1) are analyzed. The analyzed region of length 264bp, which includes HVR1, has been sequenced using the GS FLX System and the GS FLX Titanium Sequencing Kit (454 Life Sciences, Roche, Branford, CT). Obtained sequences were processed using the error correction and haplotyping algorithm KEC [122], and the obtained haplotypes were aligned using Muscle [123]. The data [102, 124] used for classification of intra-host HCV populations as recent and chronic consists of 365 NGS samples, including 108 datasets corresponding to recently infected hosts and 257 datasets belonging to chronically infected hosts. Recent samples either belong to patients with the known times since seroconversion, or to the collection of HCV outbreaks, where epidemiological investigations revealed that secondary cases were infected within few months from the dates of sample collection, thus allowing to classify them as recently infected. Chronic samples are obtained from several




(a) Recent (b) Chronic

Figure (6.3) Examples of normalized images of intra-host populations from (a) recent HCV infection and (b) chronic HCV infection.

molecular surveillance studies.

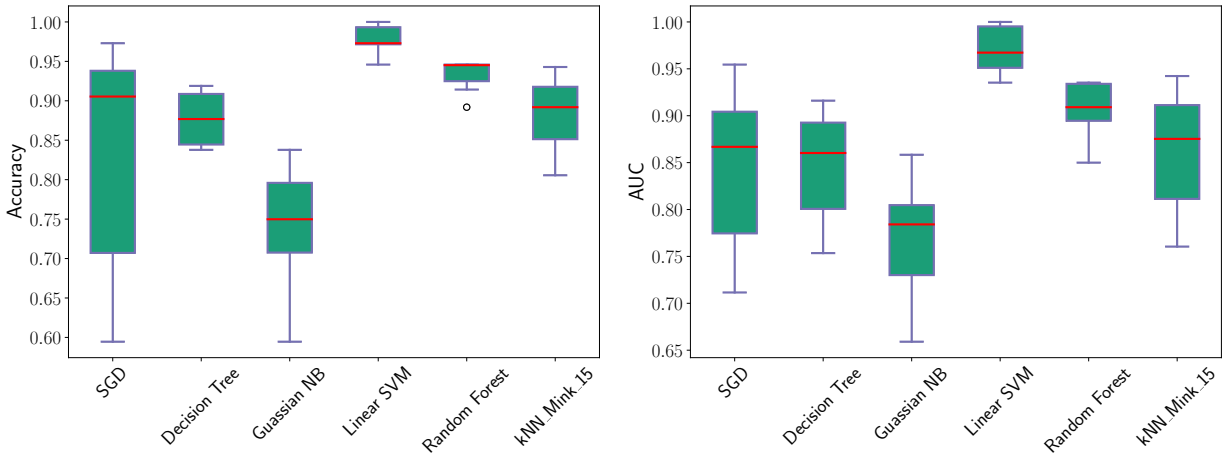
**Classification** Identification of HCV infection stages is considered as a binary classification problem. Fig.6.3 shows typical normalized images of HCV populations from recent and chronic infections. Visual inspection of images allows for identification of typical patterns associated with both classes – images of recent infection have pronounced diagonal lines while chronic images are choppy.

Images corresponding to intra-host viral populations have been labeled based on the stage of infection as recent or chronic and used to train the following machine learning classification models: Stochastic Gradient Descent (SGD) [125], decision trees [126], Gaussian Naive Bayes(Gaussian NB) [127], Linear Support Vector Machine (Linear SVM) [128], Random Forest [129] and $k$-Nearest Neighbours($k$NN) [130,131]. We used models' implementations from python `scikit-learn` library [132]. Different SVM kernels have been explored of which SVM with linear kernel produced the best results. In linear SVM model, there is a

regularization parameter $c$ which helps in generalizing the model by controlling testing and training errors. In this model, grid search is performed on $c$ values in the range $[-2, 20]$. For k-NN models, we selected the best model among the models with euclidean and manhattan metrics and with $k$ from the range $[3, 20]$. For random forest, the best model has been chosen by performing grid search on the number of trees in the range $[10, 100]$.

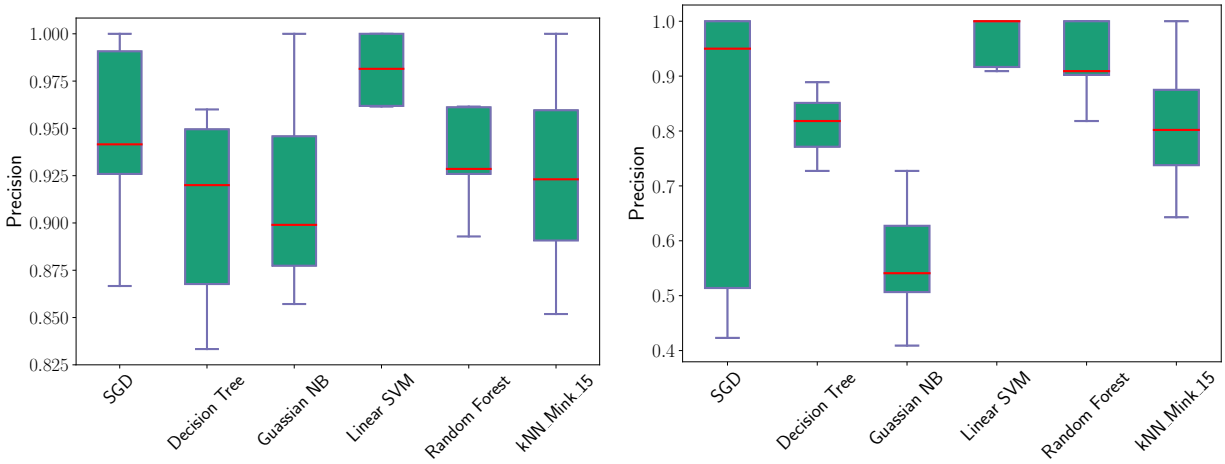
Trained classifiers have been validated based on their accuracy, area under the curve (AUC), precision, and recall. Accuracy (Acc) is defined as the proportion of test cases correctly classified as either recent or chronic. Precision (Prec) measures the fraction of the correctly classified populations within each predicted infection class, while recall (Rec) measures the fraction of the true recent or chronic populations that are correctly predicted. Validation has been performed via stratified 10-fold cross validation. Specifically, in addition to the standard 10-fold cross-validation, we employ "leave-one-outbreak-out" cross-validation and random undersampling methods to balance the datasets. In our current data, some of the samples come from the same HCV outbreak. Such samples are close to each other by their nucleotide composition, thus their presence may lead to over-fitting of any particular method. In "leave-one-outbreak-out" cross-validation, data from each of these outbreaks was used in the validation set, while other samples are used in the training sets. Random undersampling has been performed to balance the difference in sizes of datasets of recent and chronic hosts. In this method, chronic dataset size is reduced by random subsampling to match the recent dataset size.

**Results** Stratified 10-fold cross validation has been initially performed to analyze performance of several classification methods trained using the normalized image data. Fig. 6.4 shows accuracy and AUC of the best models for each of the methods using box plots, with the average metrics being indicated by red line. Linear SVM demonstrated superior performance compared to all other models, with an average accuracy of 97.545% and low accuracy variance. Other models with the exception of Gaussian NB have accuracy greater than 85%, thus exceeding accuracy of existing methods, which are primarily based on feature



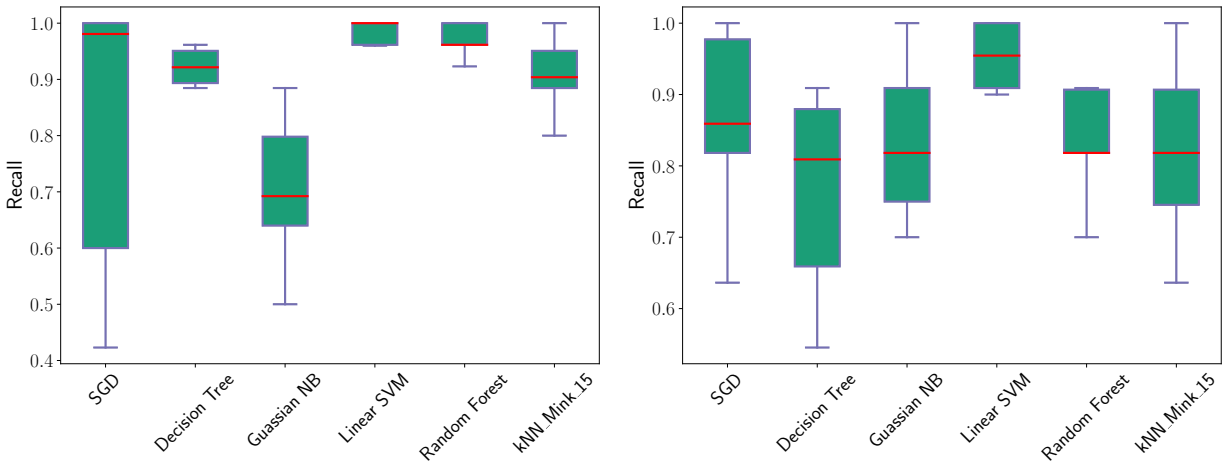

(a) Accuracy (b) AUC (Area under the curve)

Figure (6.4) Accuracy and AUC (Area Under the Curve) for several simple classification methods after training based on the normalized image data.




(a) Precision(Chronic) (b) Precision(Acute)

Figure (6.5) Precision of several simple classification methods after training based on the image data generated using the sequence image preprocessing method.




(a) Recall(Chronic)

(b) Recall(Acute)

Figure (6.6) Recall comparisons of several simple classification methods after training based on the image data generated using the sequence image preprocessing method.

extraction methods (see Comparison with previous methods subsection). Experiments have also been performed using various deep learning models including transfer learning, but these models have less accuracy ( 73%) due to very small training datasets. Custom built models and also transfer learning [133] with resnet-50 models have accuracy of 71.3% and 72% respectively. Performing gradient amplification on these models improves the accuracy further to 72.4% and 73.9% respectively.

Accuracy metric alone cannot define performance of the model as it needs to achieve higher precision and recall metrics for each infection type as well. Fig. 6.5(a) - 6.6(b) demonstrate the precision and recall metrics for chronic and recent samples separately. As before, linear SVM achieved the best performance over all other models with an average precision and recall values of 98.11% and 98.45% for chronic populations and 96.52% and 95.36% for recent populations, respectively. This model also has low variance across the values obtained from all the folds. Noticeably, other models with the exception of Gaussian NB also achieve more than 80% values for these metrics.

Linear SVM model has been analyzed further with leave-one-outbreak-out and random undersampling validation combined with 10-fold cross-validation. Table 6.1 shows the results

Table (6.1) Performance metrics of Linear SVM classifier assessed by standard 10-fold cross validation, leave-one-outbreak-out validation and random undersampling methods

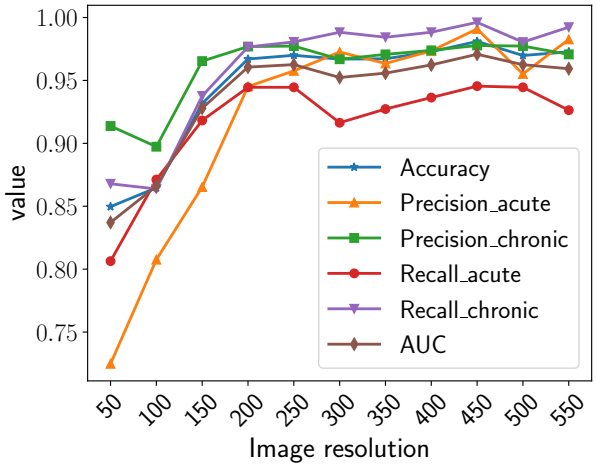
| Sampling Methods | Accuracy | Precision-Chronic | Precision-Recent | Recall-Chronic | Recall-Recent | AUC |
|---|---|---|---|---|---|---|
| Standard 10-fold cross-validation | 97.545% | 98.105% | 96.515% | 98.446 % | 95.364% | 96.905% |
| Leave-one-outbreak-out | 96.075% | 97.004% | 91.0% | 98.446 % | 83.5% | 90.973% |
| Random undersampling | 95.164% | 96.328% | 94.661% | 94.155 % | 96.173% | 95.164% |

of these methods compared to the standard 10-fold cross validation on the whole dataset. The classification accuracy remains stable under the additional sampling methods.

**Comparison with previous methods** A previously published model [113] classifies stages of HCV infection using one of the following 3 parameters: variant frequencies entropy, average position-wise nucleotide entropy and the average distance from viral variants to the most frequent variant of the population. In our data, AUC for these parameters was equal to $\sim 81\%$, $\sim 66\%$ and $\sim 78\%$, respectively, while the proposed classifier based on image normalization yielded $\sim 96.9\%$ AUC.

### 6.4.2 Effect of image resolution

All experimental results discussed above have been obtained using the default image resolution $480 \times 480$. We analyzed impact of image resolution on the classification and clustering performance. Resolution values varied from $50 \times 50$ to $550 \times 550$ with step size of 50. Fig. 6.7(a) shows the performance metrics of stratified 10-fold cross validation using LinearSVM model for detecting stage of HCV infections based on different image resolutions. Highest accuracy is achieved at the resolution 450x450, although the accuracy mostly saturates approximately after the resolution $300 \times 300$.




(a) Classification

Figure (6.7) Performance metrics (Y-axis) of classification methods based on different image resolutions(X-axis).

## 6.5 Summary & Future work

Our proposed preprocessing method converts the viral population genomic data sampled by NGS into a scaled image. Irregularities in the data thus are handled by generating a fixed size image. The number of features in this case remains same. Therefore, it can be directly used for machine learning applications without any explicit feature selection methods. High accuracy of machine learning classification technique based on image representation applied to several several models signifies validity of our approach. We plan to understand our prediction better through rule generation [134] and use other machine learning technologies such as Clustering SVM [135] to improve our results. The case of infection staging is particularly illustrative. We would like to employ deep learning models using transfer learning and gradient amplification to further improve the performance.

# 7

# CONCLUSION

In this proposal, we present gradient amplification method to train deep neural network which achieves higher accuracies even with larger learning rates and addresses vanishing gradient problem. This also improves training time compared to existing methods. We also emphasize on training strategies which play a crucial role in performance improvement when gradient amplification is applied. We suggest three approaches for gradient amplification on layers namely random selection, formulated layer detection using normalized layer gradient directionality ratio measures $G$ and $\hat{G}$. We perform detailed analysis on CIFAR-10 and CIFAR-100 datasets on simple and deeper networks and show that amplifying gradients of a few layers with a factor 2 is sufficient to improve the performance of the model. We also apply deep learning models with feed forward architectures to detect data integrity attacks in smart grids. The performance of these models is analyzed and compared with other current popular methods for a range of attacked meters. This analysis shows that feed-forward neural networks can detect attacks with better performance than attack detection algorithms that employ state vector estimation methods for centralized data integrity attacks. In bioinformatics, we propose a preprocessing method which converts the heterogenous viral population genomic data into a scaled image. Irregularities in the data thus are handled by generating a fixed size image. The number of features in this case remains same. Therefore, it can be directly used for machine learning applications without any explicit feature selection methods. High accuracy of machine learning classification and clustering techniques based on image representation applied to several molecular epidemiology tasks signifies validity of our approach.

## 7.1 Future work

### 7.1.1 Extending gradient amplification on other deep learning architectures

In this work, gradient amplification is experimented on a VGG and resnet models. This method can be easily integrated into various deep learning architectures [136] such as deep belief networks [137], recurrent neural networks [138], attention networks [139], fuzzy/hybrid neural networks [140, 141, 141–146] and graph neural networks [147].

### 7.1.2 Gradient modification including both amplification and reduction

This work analyzes the performance gains of gradient amplification approaches and establishes that modifying gradients is analogous to modifying learning rates. While training deep learning models, whenver the weights are close to optima, gradients can be reduced so that smaller learning steps are carefully taken to reach optimal solution. Computation of ratio measures in this research focuses on the gradient fluctuations across all the iterations of an epoch. Instead it can also be computed using only a few iterations of an epoch. As gradient modification dynamically changes learning rate, it can be integrated with optimization algorithms [148, 149] to perform adaptive updates.

### 7.1.3 Data driven gradient modification

Currently, we formulate ratio measures based on the mean direction of the gradient updates by analyzing how gradients of a layer modify weights while training. Gradient amplification can also be done based on data features [150]. Some of the features in the data are more prominent and have better discriminative capability compared to other features. This information can be integrated while training models, so that gradients are modified according to the importance of the features extracted in deep learning models.

# REFERENCES

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[2] K. Gopalakrishnan, S. K. Khaitan, A. Choudhary, and A. Agrawal, "Deep convolutional neural networks with transfer learning for computer vision-based data-driven pavement distress detection," *Construction and Building Materials*, vol. 157, pp. 322–330, 2017.

[3] M. D. Zeiler, "Hierarchical convolutional deep learning in computer vision," Ph.D. dissertation, New York University, 2013.

[4] R. Shanmugamani, *Deep Learning for Computer Vision: Expert techniques to train advanced neural networks using TensorFlow and Keras*. Packt Publishing Ltd, 2018.

[5] Z. Yu, T. Li, G. Luo, H. Fujita, N. Yu, and Y. Pan, "Convolutional networks with cross-layer neurons for image recognition," *Information Sciences*, vol. 433, pp. 241–254, 2018.

[6] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal processing magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[7] J. Padmanabhan and M. J. Johnson Premkumar, "Machine learning in automatic speech recognition: A survey," *IETE Technical Review*, vol. 32, no. 4, pp. 240–251, 2015.

[8] J. Huang and B. Kingsbury, "Audio-visual deep learning for noise robust speech recognition," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2013, pp. 7596–7599.

[9] N. Morgan, "Deep and wide: Multiple layers in automatic speech recognition," *Ieee transactions on audio, speech, and language processing*, vol. 20, no. 1, pp. 7–13, 2011.

[10] W. Zhang, X. Cui, U. Finkler, G. Saon, A. Kayi, A. Buyuktosunoglu, B. Kingsbury, D. Kung, and M. Picheny, "A highly efficient distributed deep learning system for automatic speech recognition," *arXiv preprint arXiv:1907.05701*, 2019.

[11] W. Zhang, X. Cui, U. Finkler, B. Kingsbury, G. Saon, D. Kung, and M. Picheny, "Distributed deep learning strategies for automatic speech recognition," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 5706–5710.

[12] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[13] D. W. Otter, J. R. Medina, and J. K. Kalita, "A survey of the usages of deep learning for natural language processing," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[14] J. Chai and A. Li, "Deep learning in natural language processing: A state-of-the-art survey," in *2019 International Conference on Machine Learning and Cybernetics (ICMLC)*. IEEE, 2019, pp. 1–6.

[15] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing," *ieee Computational intelligenCe magazine*, vol. 13, no. 3, pp. 55–75, 2018.

[16] R. Socher, "Recursive deep learning for natural language processing and computer vision," Ph.D. dissertation, Citeseer, 2014.

[17] A. Kamilaris and F. X. Prenafeta-Boldú, "Deep learning in agriculture: A survey," *Computers and electronics in agriculture*, vol. 147, pp. 70–90, 2018.

[18] G. Litjens, T. Kooi, B. E. Bejnordi, A. A. A. Setio, F. Ciompi, M. Ghafoorian, J. A. Van Der Laak, B. Van Ginneken, and C. I. Sánchez, "A survey on deep learning in medical image analysis," *Medical image analysis*, vol. 42, pp. 60–88, 2017.

[19] Q. Zhang, L. T. Yang, Z. Chen, and P. Li, "A survey on deep learning for big data," *Information Fusion*, vol. 42, pp. 146–157, 2018.

[20] L. Zhang, S. Wang, and B. Liu, "Deep learning for sentiment analysis: A survey," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 8, no. 4, p. e1253, 2018.

[21] J. Wang, Y. Chen, S. Hao, X. Peng, and L. Hu, "Deep learning for sensor-based activity recognition: A survey," *Pattern Recognition Letters*, vol. 119, pp. 3–11, 2019.

[22] M. J. Zaki and Y. Pan, "Introduction: Recent developments in parallel and distributed data mining," *Distributed and Parallel Databases*, vol. 11, no. 2, pp. 123–127, 2002.

[23] S. Bacchi, L. Oakden-Rayner, T. Zerner, T. Kleinig, S. Patel, and J. Jannes, "Deep learning natural language processing successfully predicts the cerebrovascular cause of transient ischemic attack-like presentations," *Stroke*, vol. 50, no. 3, pp. 758–760, 2019.

[24] M. Yan, L. Liu, S. Basodi, and Y. Pan, "Multi-view learning for benign epilepsy with centrotemporal spikes," *IET Computer Vision*, vol. 13, no. 2, pp. 109–116, 2018.

[25] X. Xiao, M. Yan, S. Basodi, C. Ji, and Y. Pan, "Efficient hyperparameter optimization in deep learning using a variable length genetic algorithm," *arXiv preprint arXiv:2006.12703*, 2020.

[26] A. Prieto, B. Prieto, E. M. Ortigosa, E. Ros, F. Pelayo, J. Ortega, and I. Rojas, "Neural networks: An overview of early research, current frameworks and new challenges," *Neurocomputing*, vol. 214, pp. 242–268, 2016.

[27] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "Model compression and acceleration for deep neural networks: The principles, progress, and challenges," *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 126–136, 2018.

[28] W. Rawat and Z. Wang, "Deep convolutional neural networks for image classification: A comprehensive review," *Neural computation*, vol. 29, no. 9, pp. 2352–2449, 2017.

[29] A. Canziani, A. Paszke, and E. Culurciello, "An analysis of deep neural network models for practical applications," *arXiv preprint arXiv:1605.07678*, 2016.

[30] C. Szegedy, A. Toshev, and D. Erhan, "Deep neural networks for object detection," in *Advances in neural information processing systems*, 2013, pp. 2553–2561.

[31] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, "Deep networks with stochastic depth," in *European conference on computer vision*. Springer, 2016, pp. 646–661.

[32] J. Brownlee, "Understand the Impact of Learning Rate on Neural Network Performance," 2019, [accessed 1 June 2019 ]. [Online]. Available: https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/

[33] S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber *et al.*, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," 2001.

[34] G. B. Goh, N. O. Hodas, and A. Vishnu, "Deep learning for computational chemistry," *Journal of computational chemistry*, vol. 38, no. 16, pp. 1291–1307, 2017.

[35] B. Hanin, "Which neural net architectures give rise to exploding and vanishing gradients?" in *Advances in Neural Information Processing Systems*, 2018, pp. 582–591.

[36] J. Schmidhuber, "Learning complex, extended sequences using the principle of history compression," *Neural Computation*, vol. 4, no. 2, pp. 234–242, 1992.

[37] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[38] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 315–323.

[39] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[40] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[41] B. Singh, S. De, Y. Zhang, T. Goldstein, and G. Taylor, "Layer-specific adaptive learning rates for deep networks," 2015.

[42] C. Darken, J. Chang, and J. Moody, "Learning rate schedules for faster stochastic gradient search," in *Neural Networks for Signal Processing II Proceedings of the 1992 IEEE Workshop*. IEEE, 1992, pp. 3–12.

[43] S. Lau, "Learning Rate Schedules and Adaptive Learning Rate Methods for Deep Learning," 2019, [accessed 1 June 2019 ]. [Online]. Available: https://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-2c8f433990d1

[44] S. L. Smith, P.-J. Kindermans, C. Ying, and Q. V. Le, "Don't decay the learning rate, increase the batch size," *arXiv preprint arXiv:1711.00489*, 2017.

[45] T. Salimans and D. P. Kingma, "Weight normalization: A simple reparameterization to accelerate training of deep neural networks," in *Advances in neural information processing systems*, 2016, pp. 901–909.

[46] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[47] M. D. Zeiler, "Adadelta: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[48] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, 2013.

[49] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[50] C.-C. Yu and B.-D. Liu, "A backpropagation algorithm with adaptive learning rate and momentum coefficient," in *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No. 02CH37290)*, vol. 2. IEEE, 2002, pp. 1218–1223.

[51] Y. Dauphin, H. De Vries, and Y. Bengio, "Equilibrated adaptive learning rates for non-convex optimization," in *Advances in neural information processing systems*, 2015, pp. 1504–1512.

[52] L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao, and J. Han, "On the variance of the adaptive learning rate and beyond," *arXiv preprint arXiv:1908.03265*, 2019.

[53] L. Luo, Y. Xiong, Y. Liu, and X. Sun, "Adaptive gradient methods with dynamic bound of learning rate," *arXiv preprint arXiv:1902.09843*, 2019.

[54] H. Gupta, R. Srikant, and L. Ying, "Finite-time performance bounds and adaptive learning rate selection for two time-scale reinforcement learning," in *Advances in Neural Information Processing Systems*, 2019, pp. 4704–4713.

[55] J. M. Ede and R. Beanland, "Adaptive learning rate clipping stabilizes learning," *Machine Learning: Science and Technology*, vol. 1, no. 1, p. 015011, 2020.

[56] C. Daniel, J. Taylor, and S. Nowozin, "Learning step size controllers for robust neural network training," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[57] Z. Xu, A. M. Dai, J. Kemp, and L. Metz, "Learning an adaptive learning rate schedule," 2019.

[58] Y. You, I. Gitman, and B. Ginsburg, "Large batch training of convolutional networks," 2017.

[59] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[60] H. Zhang, W. Chen, and T.-Y. Liu, "Train feedfoward neural network with layer-wise adaptive rate via approximating back-matching propagation," 2018.

[61] S. Basodi, C. Ji, H. Zhang, and Y. Pan, "Gradient amplification: An efficient way to train deep neural networks," *arXiv preprint arXiv:2006.10560*, 2020.

[62] S. Basodi, P. B. Icer, P. Skums, Y. Khudyakov, A. Zelikovsky, and Y. Pan, "Classification of hcv infections through sequence image normalization," in *2017 IEEE 7th International Conference on Computational Advances in Bio and Medical Sciences (ICCABS)*. IEEE, 2017, pp. 1–1.

[63] S. Basodi, P. I. Baykal, A. Zelikovsky, P. Skums, and Y. Pan, "Analysis of heterogeneous genomic samples using image normalization and machine learning," *bioRxiv*, p. 642108, 2019.

[64] S. Basodi, S. Tan, W. Song, and Y. Pan, "Data integrity attack detection in smart grid: a deep learning approach," *International Journal of Security and Networks*, vol. 15, no. 1, pp. 15–24, 2020.

[65] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[66] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch," in *NIPS Autodiff Workshop*, 2017.

[67] X. Xiao, T. B. Mudiyanselage, C. Ji, J. Hu, and Y. Pan, "Fast deep learning training through intelligently freezing layers," in *2019 International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE, 2019, pp. 1225–1232.

[68] F. Aloul, A. Al-Ali, R. Al-Dalky, M. Al-Mardini, and W. El-Hajj, "Smart grid security: Threats, vulnerabilities and solutions," *International Journal of Smart Grid and Clean Energy*, vol. 1, no. 1, pp. 1–6, 2012.

[69] A. R. Metke and R. L. Ekl, "Smart grid security technology," in *Innovative Smart Grid Technologies (ISGT), 2010*. IEEE, 2010, pp. 1–7.

[70] W. Wang and Z. Lu, "Cyber security in the smart grid: Survey and challenges," *Computer Networks*, vol. 57, no. 5, pp. 1344–1371, 2013.

[71] A. Abur and A. G. Exposito, *Power system state estimation: theory and implementation*. CRC press, 2004.

[72] Y. Liu, P. Ning, and M. K. Reiter, "False data injection attacks against state estimation in electric power grids," *ACM Transactions on Information and System Security (TISSEC)*, vol. 14, no. 1, p. 13, 2011.

[73] O. Kosut, L. Jia, R. J. Thomas, and L. Tong, "Malicious data attacks on smart grid state estimation: Attack strategies and countermeasures," in *Smart Grid Communica-*

*tions (SmartGridComm), 2010 First IEEE International Conference on.* IEEE, 2010, pp. 220–225.

[74] A. Giani, E. Bitar, M. Garcia, M. McQueen, P. Khargonekar, and K. Poolla, "Smart grid data integrity attacks: characterizations and countermeasures $\pi$," in *Smart Grid Communications (SmartGridComm), 2011 IEEE International Conference on.* IEEE, 2011, pp. 232–237.

[75] M. Esmalifalak, L. Liu, N. Nguyen, R. Zheng, and Z. Han, "Detecting stealthy false data injection using machine learning in smart grid," *IEEE Systems Journal*, 2014.

[76] M. Ozay, I. Esnaola, F. T. Y. Vural, S. R. Kulkarni, and H. V. Poor, "Machine learning methods for attack detection in the smart grid," *IEEE transactions on neural networks and learning systems*, vol. 27, no. 8, pp. 1773–1786, 2016.

[77] M. Ozay, I. Esnaola, F. T. Vural, S. R. Kulkarni, and H. V. Poor, "Sparse attack construction and state estimation in the smart grid: Centralized and distributed models," *IEEE Journal on Selected Areas in Communications*, vol. 31, no. 7, pp. 1306–1318, 2013.

[78] O. Bousquet, S. Boucheron, and G. Lugosi, "Introduction to statistical learning theory," in *Advanced lectures on machine learning.* Springer, 2004, pp. 169–207.

[79] PJM, "PJM: Markets and Operations ," [accessed 2017-06-23 ]. [Online]. Available: http://www.pjm.com/markets-and-operations.aspx

[80] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting." *Journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[81] R. D. Zimmerman, C. E. Murillo-Sánchez, and R. J. Thomas, "Matpower: Steady-state operations, planning, and analysis tools for power systems research and education," *IEEE Transactions on power systems*, vol. 26, no. 1, pp. 12–19, 2011.

[82] F. Chollet, "Keras," https://github.com/fchollet/keras, 2015.

[83] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/

[84] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *arXiv preprint arXiv:1511.07289*, 2015.

[85] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[86] R. Vijayanand, D. Devaraj, and B. Kannapiran, "Intrusion detection system for wireless mesh network using multiple support vector machine classifiers with genetic-algorithm-based feature selection," *Computers & Security*, vol. 77, pp. 304–314, 2018.

[87] M. Panda, "Intelligent data analysis for sustainable smart grids using hybrid classification by genetic algorithm based discretization," *Intelligent Decision Technologies*, vol. 11, no. 2, pp. 137–151, 2017.

[88] C. Bharathi, D. Rekha, and V. Vijayakumar, "Genetic algorithm based demand side management for smart grid," *Wireless Personal Communications*, vol. 93, no. 2, pp. 481–502, 2017.

[89] X. Fu, A. G. Bourgeois, P. Fan, and Y. Pan, "Using a genetic algorithm approach to solve the dynamic channel-assignment problem," *International Journal of Mobile Communications*, vol. 4, no. 3, pp. 333–353, 2006.

[90] J. He, S. Ji, M. Yan, Y. Pan, and Y. Li, "Load–balanced cds construction in wireless sensor networks via genetic algorithm," *International Journal of Sensor Networks*, vol. 11, no. 3, pp. 166–178, 2012.

[91] G. A. Mary and R. Rajarajeswari, "Smart grid cost optimization using genetic algorithm," *Int. J. Res. Eng. Technol*, vol. 3, no. 07, pp. 282–287, 2014.

[92] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "Lstm: A search space odyssey," *IEEE transactions on neural networks and learning systems*, vol. 28, no. 10, pp. 2222–2232, 2016.

[93] I. Astrovskaya, N. Mancuso, B. Tork, S. Mangul, A. Artyomenko, P. Skums, L. Ganova-Raeva, I. Măndoiu, A. Zelikovsky, and M. Park, "Inferring viral quasispecies spectra from shortgun and aplicon next-generation sequencing reads," *Genome analysis: current procedures and applications*, 2014.

[94] M. Marz, N. Beerenwinkel, C. Drosten, M. Fricke, D. Frishman, I. L. Hofacker, D. Hoffmann, M. Middendorf, T. Rattei, P. F. Stadler *et al.*, "Challenges in rna virus bioinformatics," *Bioinformatics*, vol. 30, no. 13, pp. 1793–1799, 2014.

[95] S. Bartlett, J. Wertheim, R. Bull, G. Matthews, F. Lamoury, K. Scheffler, M. Hellard, L. Maher, G. Dore, A. Lloyd *et al.*, "A molecular transmission network of recent hepatitis c infection in people with and without hiv: Implications for targeted treatment strategies," *Journal of viral hepatitis*, vol. 24, no. 5, pp. 404–411, 2017.

[96] M. G. Collier, Y. E. Khudyakov, D. Selvage, M. Adams-Cameron, E. Epson, A. Cronquist, R. H. Jervis, K. Lamba, A. C. Kimura, and R. Sowadsky, "Outbreak of hepatitis

a in the usa associated with frozen pomegranate arils imported from turkey: an epidemiological case study," *The Lancet Infectious Diseases*, vol. 14, no. 10, pp. 976–981, 2014.

[97] M. K. Grabowski and A. D. Redd, "Molecular tools for studying hiv transmission in sexual networks," *Current Opinion in HIV and AIDS*, vol. 9, no. 2, pp. 126–133, 2014.

[98] W. C. Hellinger, L. P. Bacalis, R. S. Kay, N. D. Thompson, G.-L. Xia, Y. Lin, Y. E. Khudyakov, and J. F. Perz, "Health care–associated hepatitis c virus infections attributed to narcotic diversion," *Annals of internal medicine*, vol. 156, no. 7, pp. 477–482, 2012.

[99] M. Kuroda, H. Katano, N. Nakajima, M. Tobiume, A. Ainai, T. Sekizuka, H. Hasegawa, M. Tashiro, Y. Sasaki, Y. Arakawa, and othes, "Characterization of quasispecies of pandemic 2009 influenza a virus (a/h1n1/2009) by de novo sequencing using a next-generation dna sequencer," *PLoS One*, vol. 5, no. 4, p. e10256, 2010.

[100] A. C. Seña, A. Moorman, L. Njord, R. E. Williams, J. Colborn, Y. Khudyakov, J. Drobenuic, G.-L. Xia, H. Wood, and Z. Moore, "Acute hepatitis b outbreaks in 2 skilled nursing facilities and possible sources of transmission north carolina, 2009–2010," *Infection Control*, vol. 34, no. 07, pp. 709–716, 2013.

[101] P. Skums, N. Mancuso, A. Artyomenko, B. Tork, I. Mandoiu, Y. Khudyakov, and A. Zelikovsky, "Reconstruction of viral population structure from next-generation sequencing data using multicommodity flows," *BMC bioinformatics*, vol. 14, no. Suppl 9, p. S2, 2013.

[102] J. Lara, M. Teka, and Y. Khudyakov, "Identification of recent cases of hepatitis c virus infection using physical-chemical properties of hypervariable region 1 and a radial basis function neural network classifier," *BMC genomics*, vol. 18, no. 10, p. 880, 2017.

[103] D. S. Campo, Z. Dimitrova, L. Yamasaki, P. Skums, D. T. Lau, G. Vaughan, J. C. Forbi, C.-G. Teo, and Y. Khudyakov, "Next-generation sequencing reveals large con-

nected networks of intra-host hcv variants," *BMC genomics*, vol. 15, no. Suppl 5, p. S4, 2014.

[104] P. Skums, A. Zelikovsky, R. Singh, W. Gussler, Z. Dimitrova, S. Knyazev, I. Mandric, S. Ramachandran, D. Campo, D. Jha *et al.*, "Quentin: reconstruction of disease transmissions from viral quasispecies genomic data," *Bioinformatics*, vol. 34, no. 1, pp. 163–170, 2017.

[105] O. Glebova, S. Knyazev, A. Melnyk, A. Artyomenko, Y. Khudyakov, A. Zelikovsky, and P. Skums, "Inference of genetic relatedness between viral quasispecies from sequencing data," *BMC genomics*, vol. 18, no. 10, p. 918, 2017.

[106] N. Yu, Z. Li, and Z. Yu, "Survey on encoding schemes for genomic data representation and feature learningfrom signal processing to machine learning," *Big Data Mining and Analytics*, vol. 1, no. 3, pp. 191–210, 2018.

[107] N. G. Douek DC, Kwong PD, "The rational design of an AIDS vaccine." *Cell*, vol. 124, pp. 677–681, 2006.

[108] J. Holland, J. de la Torre, and D. Steinhauer, "Rna virus populations as quasispecies," *Current Topics in Microbiology and Immunology, 176*, pp. 1–20, 1992.

[109] S.-Y. Rhee, T. F. Liu, S. P. Holmes, and R. W. Shafer, "HIV-1 subtype B protease and reverse transcriptase amino acid covariation," *PLoS Comput Biol*, vol. 3, no. 5, p. e87, May 2007. [Online]. Available: http://dx.doi.org/10.1371%2Fjournal.pcbi.0030087

[110] D. S. Campo, P. Skums, Z. Dimitrova, G. Vaughan, J. C. Forbi, C.-G. Teo, Y. Khudyakov, and D. T. Lau, "Drug resistance of a viral population and its individual intrahost variants during the first 48 hours of therapy," *Clinical Pharmacology & Therapeutics*, vol. 95, no. 6, pp. 627–635, 2014.

[111] P. Skums, L. Bunimovich, and Y. Khudyakov, "Antigenic cooperation among intrahost hcv variants organized into a complex network of cross-immunoreactivity," *Proceedings of the National Academy of Sciences*, vol. 112, no. 21, pp. 6653–6658, 2015.

[112] G. E. Fischer, M. K. Schaefer, B. J. Labus, L. Sands, P. Rowley, I. A. Azzam, P. Armour, Y. E. Khudyakov, Y. Lin, and G. Xia, "Hepatitis c virus infections from unsafe injection practices at an endoscopy clinic in las vegas, nevada, 2007–2008," *Clinical infectious diseases*, vol. 51, no. 3, pp. 267–273, 2010.

[113] V. Montoya, A. D. Olmstead, N. Z. Janjua, P. Tang, J. Grebely, D. Cook, P. Richard Harrigan, and M. Krajden, "Differentiation of acute from chronic hepatitis c virus infection by nonstructural 5b deep sequencing: A population-level tool for incidence estimation," *Hepatology*, vol. 61, no. 6, pp. 1842–1850, 2015.

[114] P. I. Baykal, A. Artyomenko, S. Ramachandran, Y. Khudyakov, A. Zelikovsky, and P. Skums, "Assessment of hcv infection stage as recent or chronic using multi-parameter analysis and machine learning," in *2017 IEEE 7th International Conference on Computational Advances in Bio and Medical Sciences (ICCABS)*. IEEE, 2017, pp. 1–1.

[115] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," *arXiv preprint arXiv:1607.01759*, 2016.

[116] S. Lai, L. Xu, K. Liu, and J. Zhao, "Recurrent convolutional neural networks for text classification." in *AAAI*, vol. 333, 2015, pp. 2267–2273.

[117] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[118] D. S. Campo, G.-L. Xia, Z. Dimitrova, Y. Lin, J. C. Forbi, L. Ganova-Raeva, L. Punkova, S. Ramachandran, H. Thai, P. Skums *et al.*, "Accurate genetic detection of hepatitis c virus transmissions in outbreak settings," *Journal of Infectious Diseases*, vol. 213, no. 6, pp. 957–965, 2016.

[119] J. O. Wertheim, A. J. L. Brown, N. L. Hepler, S. R. Mehta, D. D. Richman, D. M. Smith, and S. L. K. Pond, "The global transmission network of hiv-1," *Journal of Infectious Diseases*, vol. 209, no. 2, pp. 304–313, 2014.

[120] J. O. Wertheim, S. L. K. Pond, L. A. Forgione, S. R. Mehta, B. Murrell, S. Shah, D. M. Smith, K. Scheffler, and L. V. Torian, "Social and genetic networks of hiv-1 transmission in new york city," *PLoS pathogens*, vol. 13, no. 1, p. e1006000, 2017.

[121] D. S. Campo, G.-L. Xia, Z. Dimitrova, Y. Lin, J. C. Forbi, L. Ganova-Raeva, L. Punkova, S. Ramachandran, H. Thai, P. Skums *et al.*, "Accurate genetic detection of hepatitis c virus transmissions in outbreak settings," *The Journal of infectious diseases*, vol. 213, no. 6, pp. 957–965, 2015.

[122] P. Skums, Z. Dimitrova, D. Campo, G. Vaughan, L. Rossi, J. Forbi, J. Yokosawa, A. Zelikovsky, and Y. Khudyakov, "Efficient error correction for next-generation sequencing of viral amplicons," *BMC Bioinformatics*, vol. 13, no. Suppl 10, p. S6, 2012.

[123] R. C. Edgar, "Muscle: multiple sequence alignment with high accuracy and high throughput," *Nucleic acids research*, vol. 32, no. 5, pp. 1792–1797, 2004.

[124] I. V. Astrakhantseva, D. S. Campo, A. Araujo, C.-G. Teo, Y. Khudyakov, and S. Kamili, "Differences in variability of hypervariable region 1 of hepatitis c virus (hcv) between acute and chronic stages of hcv infection," *In silico biology*, vol. 11, no. 5, pp. 163–173, 2011.

[125] B. Zadrozny and C. Elkan, "Transforming classifier scores into accurate multiclass probability estimates," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2002, pp. 694–699.

[126] S. R. Safavian and D. Landgrebe, "A survey of decision tree classifier methodology," *IEEE transactions on systems, man, and cybernetics*, vol. 21, no. 3, pp. 660–674, 1991.

[127] I. Rish *et al.*, "An empirical study of the naive bayes classifier," in *IJCAI 2001 workshop on empirical methods in artificial intelligence*, vol. 3, no. 22, 2001, pp. 41–46.

[128] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[129] A. Liaw, M. Wiener *et al.*, "Classification and regression by randomforest," *R news*, vol. 2, no. 3, pp. 18–22, 2002.

[130] N. S. Altman, "An introduction to kernel and nearest-neighbor nonparametric regression," *The American Statistician*, vol. 46, no. 3, pp. 175–185, 1992.

[131] L. Jiang, Z. Cai, D. Wang, and S. Jiang, "Survey of improving k-nearest-neighbor for classification," in *Fourth international conference on fuzzy systems and knowledge discovery (FSKD 2007)*, vol. 1. IEEE, 2007, pp. 679–683.

[132] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[133] L. Torrey and J. Shavlik, "Transfer learning," in *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*. IGI global, 2010, pp. 242–264.

[134] J. He, H.-J. Hu, R. Harrison, P. C. Tai, and Y. Pan, "Rule generation for protein secondary structure prediction with support vector machines and decision tree," *IEEE Transactions on nanobioscience*, vol. 5, no. 1, pp. 46–53, 2006.

[135] W. Zhong, J. He, R. Harrison, P. C. Tai, and Y. Pan, "Clustering support vector machines for protein local structure prediction," *Expert Systems with Applications*, vol. 32, no. 2, pp. 518–526, 2007.

[136] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, "A survey of deep neural network architectures and their applications," *Neurocomputing*, vol. 234, pp. 11–26, 2017.

[137] N. Le Roux and Y. Bengio, "Representational power of restricted boltzmann machines and deep belief networks," *Neural computation*, vol. 20, no. 6, pp. 1631–1649, 2008.

[138] B. A. Pearlmutter, "Gradient calculations for dynamic recurrent neural networks: A survey," *IEEE Transactions on Neural networks*, vol. 6, no. 5, pp. 1212–1228, 1995.

[139] S. Chaudhari, G. Polatkan, R. Ramanath, and V. Mithal, "An attentive survey of attention models," *arXiv preprint arXiv:1904.02874*, 2019.

[140] H. Liu, J. Li, Y.-Q. Zhang, and Y. Pan, "An adaptive genetic fuzzy multi-path routing protocol for wireless ad-hoc networks," in *Sixth International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing and First ACIS International Workshop on Self-Assembling Wireless Network.* IEEE, 2005, pp. 468–475.

[141] J. Zhang, J.-S. Wong, T. Li, and Y. Pan, "A comparison of parallel large-scale knowledge acquisition using rough set theory on different mapreduce runtime systems," *International Journal of Approximate Reasoning*, vol. 55, no. 3, pp. 896–907, 2014.

[142] J. Zhang, T. Li, and Y. Pan, "Parallel rough set based knowledge acquisition using mapreduce from big data," in *Proceedings of the 1st International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications*, 2012, pp. 20–27.

[143] T. K. B. Mudiyanselage, X. Xiao, Y. Zhang, and Y. Pan, "Deep fuzzy neural networks for biomarker selection for accurate cancer detection," *IEEE Transactions on Fuzzy Systems*, 2019.

[144] J. Zhang, J.-S. Wong, Y. Pan, and T. Li, "A parallel matrix-based method for computing approximations in incomplete information systems," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 2, pp. 326–339, 2014.

[145] J. Zhang, Y. Zhu, Y. Pan, and T. Li, "Efficient parallel boolean matrix based algorithms for computing composite rough set approximations," *Information Sciences*, vol. 329, pp. 287–302, 2016.

[146] S. Zhou, Q. Chen, and X. Wang, "Fuzzy deep belief networks for semi-supervised sentiment classification," *Neurocomputing*, vol. 131, pp. 312–322, 2014.

[147] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[148] Z. Wang, H. Xing, T. Li, Y. Yang, R. Qu, and Y. Pan, "A modified ant colony optimization algorithm for network coding resource minimization," *IEEE Transactions on evolutionary computation*, vol. 20, no. 3, pp. 325–342, 2015.

[149] X. Wang, Q. Li, N. Xiong, and Y. Pan, "Ant colony optimization-based location-aware routing for wireless sensor networks," in *International conference on wireless algorithms, systems, and applications*. Springer, 2008, pp. 109–120.

[150] L. Wang, Y. Yang, R. Min, and S. Chakradhar, "Accelerating deep neural network training with inconsistent stochastic gradient descent," *Neural Networks*, vol. 93, pp. 219–229, 2017.